TURBO DEBUGGER®
für Windows

BENUTZER-
HANDBUCH

BORLAND

Turbo Debugger für Windows

Benutzerhandbuch

BORLAND

Copyright © 1991 by Borland International, INC.
Jede Vervielfältigung dieses Handbuchs sowie des Softwareprogramms Turbo Debugger für Windows wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und die Rechte an dem Softwareprogramm Turbo Debugger für Windows liegen bei Borland International.

Der rechtmäßige Erwerb der Programmdisketten und der Handbücher erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daßein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Turbo Debugger für Windows nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.

Die Beispielprogramme zum Turbo Debugger für Windows, die auf den Originaldisketten mitgeliefert werden, demonstrieren die verschiedenen Eigenschaften des Turbo Debugger für Windows. Sie dienen ausschließlich zu Lehrzwecken. Sie dürfen diese Beispielprogramme editieren oder für den eigenen Gebrauch modifizieren. Sie dürfen diese Programme jedoch nicht als Ganzes oder in Teilen in compilierter Form oder als Quelltext weitergeben oder verkaufen. Sie dürfen die Routinen aus den Beispielprogrammen in eigenen Anwenderprogrammen verwenden, solange die resultierenden Programme nicht einem Beispielprogramm oder einem Teil eines Beispielprogramms in Erscheinung oder Funktion gleichen.

Einschränkung der Gewährleistung
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber
Borland GmbH
Lindwurmstraße 88
Postfach 15 03 40
8000 München 2
Telefon 0 89 /7 20 10-0
Telefax 0 89 / 77 93 39
Telex 5 212 637 mcm d

Übertragung ins Deutsche:
Marcus O. Lerch, Alois Stockklauser und Systec GmbH, Nürnberg

Typographische Konzeption des Innenteils: Rudolf Paulus Gorbach

Grafiken: Sid Gastl

PostScript-Belichtung; CD Computer & Dokumentations GmbH, Aalen

Druck und Bindung: Schoder Druck GmbH, Gersthofen

Papier: Schleipen GmbH, Bad Dürkheim. Das verwendete Papier ist säurefrei und alterungsbeständig. Die Produktion erfolgt in einem geschlossenen Wasserkreislauf ohne Einsatz von optischen Aufhellern und unter Verwendung von chlorarm gebleichten, holzfreien Zellstoffqualitäten. Die Abwasserreinigung wird in einer betriebseigenen chemisch-biologischen Kläranlage vorgenommen.

April 1991

Inhalt

Einführung 1

Voraussetzungen an Hardware und Software 1
Eine Bemerkung zur Terminologie 2
Übersicht über dieses Handbuch 2
Copyright-Erklärung 4
Warenzeichen 4
Haben Sie Fragen oder Probleme?
Die technische Beratung von Borland 4
Empfehlenswerte Bücher 5
Bücher über Turbo Debugger und seine Tools 5
Bücher über MS-Windows 5

1. Disketten und Installation 7

Die Originaldisketten 7
Die Datei README 8
Die Datei HELPME!.TDW 8
Die Datei MANUAL.TDW 8
Dienstprogramme für TDW 8
Wie geht's weiter? 9
Neulinge im Umgang mit Turbo Pascal 9
Fortgeschritten im Umgang mit Turbo Pascal, aber Neuling mit TDW 10
Fortgeschrittene im Umgang mit Turbo Debugger 10

2. Fehlersuche mit Turbo Debugger 11

Was ist Debuggen? 11
Liegt überhaupt ein Fehler vor? 12
Wo ist der Fehler? 12
Um was für eine Art von Fehler handelt es sich? 12
Wie kann man den Fehler beheben? 12
Wozu Sie Turbo Debugger einsetzen können 13
Was Turbo Debugger nicht kann 14
Wie Turbo Debugger vorgeht 15
Die Vorzüge von Turbo Debugger 15

Menüs und Dialogfenster 15
Gewußt wo 18
Geschichtsstunden 20
Sukzessive Auswahl 21
Makros erstellen 22
Schaufensterbummel 22
Das Hilfesystem 31

3. Einführung in das Debuggen 35

Das Beispielprogramm 35
Turbo Debugger im Einsatz 37
Die Menüleiste 37
Die Statuszeile 37
Die Fenster 38
Das Beispielprogramm TDDEMO 39
Breakpoints 40
Das Watches-Fenster 41
Einfache Datenobjekte in Pascal untersuchen 42
Zusammengesetzte Datenstrukturen in Pascal untersuchen 43
Pascal-Daten ändern 44

4. Der Aufruf von Turbo Debugger 47

Die Vorbereitung des Programms für das Debuggen 47
Den Debugger starten 48
Der Kommandozeilenaufruf von Turbo Debugger 48
Kommandozeilenoptionen direkt eingeben 49
Kommandozeilenoptionen in TPW.INI angeben 49
Zur Wiederholung 50
Turbo Debugger und Windows 50
Kommandozeilen-Optionen 51
Laden der Konfigurationsdatei (-c) 51
Bildschirmumschaltung (-d) 51
Hilfestellungen (-h und -?) 52
Assembler-Modus (-l) 52
Maus-Unterstützung (-p) 52
Quellcode-Optionen (-s) 53
Startverzeichnis (-t) 53
Konfigurations-Dateien 54
Das Menü Options 54
Das Auswahlfeld Language 54
Das Menü Macros 54
Das Dialogfenster Display options 55
Der Befehl Path for source 57
Der Befehl Save options 57

Der Befehl Restore options 58
Rückkehr zu Windows 58

5. Steuerung der Programmausführung 59

Den aktuellen Zustand des Programms untersuchen 60
Das Variables-Fenster 60
Das Fenster Stack 63
Der Befehl Origin im lokalen Menü 64
Der Befehl Get Info 65
Status line messages 65
Das Menü Run 67
Run F9 67
Go to cursor F4 68
Trace into F7 68
Step over F8 68
Execute to... [Alt][F9] 69
Until return [Alt][F8] 69
Animate... 69
Back trace 69
Instruction trace [Alt][F7] 70
Arguments... 70
Program reset [Ctrl][F2] 70
Das Fenster Execution History 71
Der Ausschnitt mit den ausgeführten Programmschritten 71
Programmausführung unterbrechen 73
Ctrl-Break 73
Programm beenden 73
Eine Debuggersitzung erneut starten 74
Programm erneut laden 74
Ein neues Programm zum Debuggen laden 75
Die Argumente eines Programms verändern 75

6. Programmdaten untersuchen und ändern 77

Das Menü Data 78
Inspect... 78
Evaluate/Modify... 78
Add Watch... 80
Function Return 80
Auf Datenobjekte in Quelldateien zeigen 81
Das Fenster Watches 81
Das lokale Menü des Watches-Fensters 82
Inspector-Fenster 83
Inspector-Fenster für Pascal-Daten 84
Inspector-Fenster für Assemblerdaten 86
Das lokale Menü des Inspector-Fensters 89

Range... 89
Change... 89
Inspect 89
Descend 90
New expression... 90
Type Cast... 90

7. Breakpoints 91

Das Menü Breakpoints 92
Toggle 92
At... 92
Changed memory global 93
Expression true global... 93
Delete all 93
Der Gültigkeitsbereich von Breakpoint-Ausdrücken 93
Das Fenster Breakpoints 94
Das lokale Menü des Breakpoint-Fensters 94
Das Fenster Log 97
Das lokale Menü des Log-Fensters 98
Einfache Breakpoints 99
Bedingte Breakpoints und der Durchlaufzähler 100
Globale Breakpoints 101
Unterbrechung bei Datenveränderungen 102
Variablenwerte protokollieren 102
Die Auswertung von Ausdrücken 102

8. Dateien untersuchen und ändern 103

Programmquellen untersuchen 103
Das Module-Fenster 104
Das lokale Menü des Module-Fensters 105
Andere Dateien untersuchen 107
Das File-Fenster 107
Das lokale Menü des File-Fensters 108

9. Ausdrücke 111

Die Auswahl der Sprache für die Auswertung der Ausdrücke 111
Adressen im Code, Adressen von Daten, Zeilennummern 112
Der Zugriff auf Symbole außerhalb des Gültigkeitsbereichs 113
Die Vorgabe eines Gültigkeitsbereichs 113
Der implizite Gültigkeitsbereich für die Auswertung von Ausdrücken 115
Byte-Folgen 115
Pascal-Ausdrücke 116
Pascal- Symbole 116
Konstanten und Zahlenformate in Pascal 117

Pascal- Strings 117
Operatoren in Pascal 117
Der Aufruf von Funktionen und Prozeduren in Pascal 118
Assembler-Ausdrücke 118
Symbole in Assembler 119
Assemblerkonstanten 119
Operatoren in Assembler 119
Die Formatsteuerung 120

10. Debuggen inobjektorientiertem Pascal 123

Das Fenster Hierarchy 123
Die Listenausschnitt mit den Objekttypen 124
Der Ausschnitt mit dem Objektbaum 124
Das lokale Menü des Hierarchiebaums 125
Das Inspector-Fenster für Objekttypen/Klassen 125
Die lokalen Menüs des Inspector-Fensters für Objekttypen/Klassen 126
Inspector-Fenster für Objekt-/Klasseninstanzen 127
Die lokalen Menüs des Inspector-Fensters für Objekt- und Klasseninstanzen 128
Der mittlere und der untere Ausschnitt 129

11. Debuggen unter Windows 131

Spezialitäten für Windows 131
Protokollieren von Windows-Botschaften 132
Erstellen von Speicher- und Modullisten 135
Debuggen von dynamischen Linkbibliotheken (DLLs) 138
Umwandlung von Speicher-Handles in Adressen 143

12. Debuggen auf Assemblerebene 145

Grenzen des Debuggens auf Quellcode-Ebene 145
Das CPU-Fenster 146
Der Code-Ausschnitt 147
Der Disassembler 148
Das lokale Menü des Code-Ausschnitts 148
Die Register- und Statusausschnitte 152
Das lokale Menü des Registerausschnitts 152
Das lokale Menü des Statusausschnitts 153
Der Datenausschnitt 153
Das lokale Menü des Datenausschnitts 154
Der Stack-Ausschnitt 157
Das lokale Menü des Stack-Ausschnitts 157
Der Assembler 158
Vorgaben für die Größe von Adreßoperanden 159
Vorgaben für den Datentyp eines Operanden 159

Stringbefehle 160
Das Fenster Dump 160
Das Fenster Registers 161

13. 80x87-Koprozessor und Gleitkomma-Emulator 163

Der 80x87-Chip im Vergleich zum Emulator 163
Das Fenster Numeric Processor 163
Der Registerausschnitt 164
Der Kontrollausschnitt 165
Der Statusausschnitt 166

14. Die Befehlsübersicht 167

Hotkeys 167
Die Befehle in der Menüleiste 169
Das Menü System 169
Das Menü File 169
Das Menü View 169
Das Menü Run 170
Das Menü Breakpoints 170
Das Menü Data 171
Das Menü Options 171
Das Fenster Window 171
Das Menü Help 172
Die Befehle in den lokalen Menüs 172
Das lokale Menü des Breakpoints-Fensters 172
Die Menüs des CPU-Fensters 173
Das Fenster Dump 175
Das Menü des Fensters Execution History 175
Das Menü des Fensters File 175
Das Menü des Log-Fensters 175
Das Menü des Module-Fensters 176
Die Menüs des Fensters Windows Messages 176
Die Menüs des Fensters Numeric processor 177
Das Fenster Hierarchy 178
Das Menü Registers 178
Das Menü des Stack-Fensters 178
Die Menüs des Fensters Variables 178
Das Menü des Fensters Watches 179
Das Menü des Inspector-Fensters 179
Das Inspector-Fenster für Objekttypen/Klassen 180
Das Inspector-Fenster für Objekt-/Klasseninstanzen 180
Textausschnitte 181
Listenausschnitte 181
Funktionstasten in Dialogfenstern und Eingabefeldern 182
Befehle zumVerändern der Fensteranordnung 182

Jokerzeichen 183
Der vollständige Menübaum 183

15. Ein Standard-Pascal-Programm debuggen 187

Wenn's nicht läuft 187
Der persönliche Stil beim Debuggen 188
Einfach mal laufen lassen 188
Schrittweise testen 188
Mögliche Fehler 189
Allgemeine Fehler 189
Typische Fehler in Pascal 191
Typische Assemblerfehler 197
Richtigkeit überprüfen 201
Grenzfälle und Einschränkungen 202
Falsche Eingabedaten 202
Leere Eingabe 202
Die Fehlersuche als Teil der Programmentwicklung 202
Beispiel für eine Debuggersitzung 203
Fehler suchen 203
Strategie festlegen 204
Der Aufruf von Turbo Debugger 205
Wandern durchs Programm 205
Das Dialogfenster Evaluate/Modify 206
Untersuchungen 207
Beobachtungen 208
Das Ende 208

16. Ein ObjectWindows-Programm debuggen 209

Über das Programm 209
Die Definition des Fenstertyps ScribbleWindow 210
Hinzufügen von Farben mit CScribbleWindow 211
Erstellen des Programms 213
Debuggen des Programms 213
Suchen des ersten Fehlers 213
Den Farbzuordnungsfehler lokalisieren 216
Lokalisieren des Fehlers 219
Den Fehler lokalisieren 221
Analyse der Fehlerquelle 222

17. Debuggen von Windows-Anwendungen 225

Die Beispielprogramme 225
Compilieren und Linken der Demoprogramme 226
Das Programm TDWDEMO starten 226
Debuggen von TDWDEMOA 226

Was ist zu tun? 227
Protokollieren von Botschaften 227
Analysieren des Botschaftenprotokolls 228
Finden des Fehlers 228
Beenden von TDWDEMOA 231
Debuggen von TDWDEMOB 232
Das Programm anhalten 233
Testen des Programms 233
Was ist zu tun? 233
Vergleichen der Speicher-Belegungslisten 234
Auffinden des Fehlers, ein funktionaler Lösungsansatz 235

A. Kommandozeilenoptionen 239

B. Inline-Assemblerbefehle 241

C. Die Installation von TDW 245
Der Aufruf von TDWINST 245
Die Einstellung der Bildschirmfarben 246
Farben anpassen 246
Die Standardfarben 248
Die Einstellung der Bildschirmparameter in Turbo Debugger 248
Display swapping 248
Integer format 249
Beginning Display 249
Screen lines 249
Tab size 250
Maximum tiled watch 250
Fast screen update 250
Permit 43/50 line mode 250
User screen updating 250
Log list length 251
Floating precision 251
Range inspect 251
Die Einstellung der Turbo-Debugger-Optionen 252
Directories 252
Input & Prompting... 252
Source Debugging... 253
Das Einstellen des Bildschirmmodus 254
Kommandozeilenoptionen und Installationsoptionen 254
Wenn alles wie gewünscht eingestellt wurde 255
Speichern der Änderungen 255
TDWINST verlassen 256

D. Dialogfenster und Fehlermeldungen 257

Dialogfenster 257
Fehlermeldungen 265
Schwerwiegende Fehlermeldungen 265
Andere Fehlermeldungen 266

E. Der Einsatz mit verschiedenen Sprachen 285

Tips für Turbo Assembler 285
Reine Hexadezimaldaten untersuchen 285
Quellcode debuggen 285
Register untersuchen und verändern 286
Tips für Turbo Pascal 286
Der Initialisierungscode 286
Die Exit-Prozeduren 287
Konstanten 287
Strings und Mengen auf dem Stack 287
Geschickte Datentypumwandlungen 288
Tips für das CPU-Fenster in Pascal 288

F. Glossar 289

Index 293

Einführung

Turbo Debugger für Windows (TDW) ist ein Quelltext-Debugger, der für Programmierer entwickelt wurde, die mit Turbo Pascal für Windows arbeiten.

Einander überlappende Fenster, Pull-Down- und Pop-Up-Menüs und umfassende Mausunterstützung sind einige der Kennzeichen der komfortablen Benutzeroberfläche. Ein kontextbezogenes Hilfesystem begleitet Sie bei allen Operationen. Einige der Eigenschaften von TDW:

- wertet alle Pascalausdrücke aus
- wertet alle Assemblerausdrücke aus
- frei konfigurierbarer Bildschirmaufbau
- voller Zugriff auf Assembler und CPU
- mächtige Breakpoint-Befehle mit Protokollfunktion
- Rückwärts-Ausführung
- Unterstützung der objektorientierten Programmierung in Turbo Pascal für Windows
- Debuggen von MS-Windows-Anwendungen
- arbeitet im Textmodus.

Voraussetzungen an Hardware und Software

Turbo Debugger stellt dieselben Anforderungen an Hard- und Software, wie Turbo Pascal für Windows auch. TDW läuft auf allen Computern, auf denen MS-Windows 3.0 oder eine neuere Version lauffähig ist.

TDW ist eine sogenannte Vollbildschirm-Anwendung, belegt also immer den ganzen Bildschirm und nicht nur einen Ausschnitt, wie andere Windows-Anwendungen. Dennoch ist TDW ein Windows-Programm und nur in dieser Umgebung lauffähig.

Ein Programm, das Sie Turbo Debugger debuggen wollen, muß eine ausführ-

bare .EXE-Datei sein, die Sie mit allen Debuggerinformationen compiliert haben.

Beachten Sie bitte, daß Sie zum Starten des Debuggers sowohl die ».EXE«-Datei als auch die Quellcodedateien benötigen. Turbo Debugger sucht die Quellcodedateien zunächst in dem Verzeichnis, in dem sie auch vom Compiler während der Compilierung gefunden wurden, anschließend in den Verzeichnissen, die im Befehl *Options/Path for source* angegeben wurden, dann im aktuellen Verzeichnis und schließlich in dem Verzeichnis, in dem die ».EXE«-Datei gespeichert ist.

Eine Bemerkung zur Terminologie

In diesem Handbuch werden manche Begriffe mit einer etwas allgemeineren Bedeutung verwendet. Dazu zählen Begriffe wie Modul und Argument.

Modul Ein Modul wird in diesem Handbuch im Sinne eines Assemblermoduls verwendet, umfaßt aber auch die Unit in Pascal.

Argument Der Begriff *Argument* ist, so wie er in diesem Handbuch verwendet wird, austauschbar mit dem Begriff Parameter. Er bezieht sich sowohl auf Kommandozeilenargumente als auch auf Argumente von Prozeduren.

Übersicht über dieses Handbuch

Hier finden Sie eine kurze Zusammenfassung der Kapitel und Anhänge in diesem Handbuch:

Kapitel 1: *Disketten und Installation* enthält eine Aufstellung aller ausgelieferten Dateien auf den Disketten und eine Beschreibung der Installation von Turbo Debugger, sowie einen kurzen nach Ihren Interessen aufgegliederten Wegweiser durch dieses Handbuch.

Kapitel 2: *Fehlersuche mit Turbo Debugger* erläutert die Benutzeroberfläche, die Menüs und die Fenstertechnik von Turbo Debugger. Des weiteren wird Ihnen gezeigt, wie Eingabeaufforderungen und Fehlermeldungen zu beantworten sind.

Kapitel 3: *Einführung in das Debuggen* erklärt zunächst einige wichtige Befehle, die daraufhin anhand eines Beispielprogramms in Turbo Pascal ausführlich erläutert werden.

Kapitel 4: *Der Aufruf von Turbo Debugger* enthält eine detaillierte Beschreibung der Kommandozeilenoptionen sowie der Konfigurationsdateien.

Kapitel 5: *Steuerung der Programmausführung* ist eine Beschreibung der *Run*-Befehle.

Kapitel 6: *Programmdaten untersuchen und ändern* beschreibt das einzigartige Konzept des *Inspector*-Fensters.

Kapitel 7: *Breakpoints* enthält eine umfassende Behandlung der Möglichkeiten mit den Turbo-Debugger-Breakpoints.

Kapitel 8: *Dateien untersuchen und ändern* beschreibt, wie man Programm- und sonstige Dateien untersuchen und ändern kann.

Kapitel 9: *Ausdrücke* enthält eine vollständige Beschreibung der Syntax der einzelnen Sprachen (Pascal, Assembler), die in Turbo Debugger verwendet werden darf.

Kapitel 10: *Debuggen in objektorientiertem Pascal* erklärt die speziellen Funktionen des Debuggers für die Untersuchung von Programmen, die in Turbo Pascal für Windows erstellt wurden.

Kapitel 11: *Debuggen unter Windows* beschreibt die speziellen Fähigkeiten von TDW, die das Debuggen von Windows-Spezifika (zum Beispiel dynamische Linkbibliotheken) erleichtern.

Kapitel 12: *Debuggen auf Assemblerebene* beschäftigt sich mit den traditionellen Eigenschaften eines Debuggers. Im Mittelpunkt steht die Erläutertung des *CPU*-Fensters.

Kapitel 13: *80x87 und Gleitkomma-Emulator* behandelt die Möglichkeiten des Numerik-Fensters, um Gleitkomma-Operationen zu debuggen.

Kapitel 14: *Die Befehlsübersicht* enthält eine tabellarische Zusammenfassung der Befehle in Turbo Debugger.

Kapitel 15: *Ein Standard-Pascal-Programm debuggen* stellt zunächst die wichtigsten Fehlerquellen in der Programmierung vor und zeigt dann anhand eines Praxisbeispiels, wie man diesen Fehlern auf die Schliche kommen kann.

Kapitel 16: *Ein ObjectWindows-Programm debuggen* zeigt, ebenfalls anhand eines Beispielprogramms, wie Fehler in einem Programm zu finden sind, das die Objekte der *ObjectWindows*-Bibliothek einbindet.

Kapitel 17: *Debuggen von Windows-Anwendungen* zeigt anhand eines Beispiels das Debuggen von Windows-Programmen.

Anhang A: *Kommandozeilenoptionen* enthält eine Aufstellung mit allen Optionen, die beim Aufruf von Turbo Debugger verwendet werden können.

Anhang B: *Inline-Assemblerbefehle* faßt die Befehle und reservierten Wörter des integrierten Assemblers zusammen.

Anhang C: *Die Installation von Turbo Debugger* beschreibt alle Optionen, die mit dem Installationsprogramm TDWINST eingestellt werden können.

Anhang D: *Dialogfenster und Fehlermeldungen* listet alle Ausgaben von Fehlermeldungen bis hin zu Eingabeaufforderungen vollständig auf.

Anhang E: *Der Einsatz mit verschiedenen Sprachen* gibt einige Tips, wie man Turbo Debugger am effizientesten mit den einzelnen Programmiersprachen einsetzt.

Anhang F: *Glossar* erklärt die wichtigsten Fachausdrücke.

Copyright-Erklärung

Jede Vervielfältigung dieses Handbuchs sowie des Turbo-Debugger-Softwareprogrammes wird strafrechtlich verfolgt. Die Rechte an dem Buch und die Rechte an dem Softwareprogramm Turbo Debugger für Windows liegen bei Borland International.

Der rechtmäßige Erwerb der Programmdisketten und der Dokumentation erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Turbo Debugger nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.

Warenzeichen

Wir nehmen in diesem Buch auf geschützte Warenzeichen Bezug, die innerhalb des laufenden Textes nicht mehr explizit als solche gekennzeichnet sind – aus dem Fehlen einer Kennzeichnung kann also nicht geschlossen werden, daß der entsprechende Produktname frei von Rechten Dritter ist:

- IBM, AT, PS/2 sind geschützte Warenzeichen von International Business Machines.
- Microsoft und MS-DOS sind geschützte Warenzeichen von Microsoft.

Haben Sie Fragen oder Probleme? Die technische Beratung von Borland

Bitte lesen Sie die Handbücher und die Datei README sorgfältig durch. Wenn danach Fragen offen geblieben sind und Sie technische Beratung brauchen, sollten Sie die folgenden Informationen parat haben:

1. Das deutsche Handbuch.

2. den Produktnamen und die Versionsnummer. (Die Versionsnummer erscheint, wenn Sie Turbo Debugger ohne Optionen aufrufen oder *About* im Menü *System* auswählen.)

3. Produktnamen, Modellbezeichnung Ihres Computers und gegebenenfalls der Zusatz-Hardware.

4. Betriebssystem und dessen Versionsnummer.

Technische Beratung erhalten Sie:

1. bei Ihrem Softwarehändler oder einem Benutzerclub;
2. bei schriftlichen Anfragen durch unsere technische Beratung;
3. in dringenden Fällen telefonisch über die Borland Hotline:
   Montag - Donnerstag 14.00 – 17.00 Uhr, Freitag 14.00 – 16.00 Uhr
   Telefon (089) 7258001

Empfehlenswerte Bücher

Viele Verlage unterstützen die Borland-Produkte durch eine Reihe exzellenter Bücher, die sowohl für Programmierneulinge als auch fortgeschrittene Anwender von Nutzen sind. Darüber hinaus sollten Sie noch Bücher über andere Themen, wie z.B. die Programmierung für Windows, lesen.

Bücher über Turbo Debugger und seine Tools

Nachstehend finden Sie eine Liste von Titeln, die zusätzliche Informationen über Turbo Debugger und seine Tools bieten:

Ackermann, Charles. *Turbo Debugger and Tools: A Self-Teaching Guide*, John Wiley and Sons (New York: 1990).

Swan, Tom. *Mastering Turbo Assembler,* Howard W. Sams and Co. (Carmel, IN: 1989).

Swan, Tom. *Mastering Turbo Debugger and Tools,* Howard W. Sams and Co. (Carmel, IN: 1990)

Syck, Gary. *The Waite Group's Turbo Assembler Bible,* Howard W. Sams and Co. (Carmel, IN: 1990).

Bücher über MS-Windows

Die folgenden Bücher bieten Informationen über das Schreiben von Windows-Anwendungsprogrammen.

Microsoft staff. *Microsoft Windows Software Development Kit, Guide to Programming,* Microsoft Corporation. (Redmond, WA: 1990).

Microsoft staff. *Microsoft Windows Software Development Kit, Reference, Vols. 1 and 2,* Microsoft Corporation. (Redmond, WA: 1990).

Microsoft staff. *Microsoft Windows Software Development Kit, Tools,* Microsoft Corporation. (Redmond, WA: 1990).

Petzold, Charles. *Programming Windows,* Microsoft Press. (Redmond, WA: 1990).

1. Disketten und Installation

Turbo Debugger für Windows (TDW) ist Bestandteil des Programmpaketes Turbo Pascal für Windows. Die ausgelieferten Disketten enthalten alle Programme, Dateien und Hilfsprogramme, die Sie für die Fehlersuche in Programmen benötigen, die in Turbo Pascal geschrieben wurden. Diese Disketten enthalten auch Dokumentationen zu Themen, die in diesem Handbuch nicht behandelt wurden. Diese finden Sie in den Dateien README und HELPME!.TDW.

Sie finden im Handbuch eine Einführung über den Leistungsumfang von TDW sowie eine vollständige Befehlsübersicht.

Bitte machen Sie, bevor Sie anfangen, mit Turbo Debugger zu arbeiten, eine Arbeitskopie der ausgelieferten Disketten und verwahren Sie diese Originaldisketten an einem sicheren Ort. Benützen Sie die Originaldisketten nur als Sicherung und erledigen Sie Ihre Arbeit mit Turbo Debugger mit Hilfe der Kopie. Die Originaldisketten sind Ihre einzige Sicherung, falls Ihre Dateien verloren gehen sollten.

Falls Sie mit der Lizenzvereinbarung von Borland noch nicht vertraut sind, sollten Sie jetzt die entsprechende Stelle in der Einleitung lesen und Ihre Registrationskarte ausfüllen und abschicken. (Sie werden dann immer automatisch informiert, wenn neue Versionen und Produkte erscheinen.)

Die Originaldisketten

Wenn Sie Turbo Pascal für Windows auf Ihrem System installieren, kopieren Sie Dateien von den Originaldisketten auf Ihre Festplatte. Das erledigt das Programm INSTALL.EXE, das Sie auf den Originaldisketten finden, automatisch. Es baut auch unter dem Programm-Manager von Windows eine Programmgruppe auf und sorgt dafür, daß in dieser Gruppe die Symbole für Turbo Pascal und TDW erscheinen.

Nach der Installation finden Sie im Verzeichnis \TPW\DOC in der Datei README eine Beschreibung der Dateien auf den Originaldisketten.

Die Datei README

Es ist sehr wichtig, daß Sie sich die Zeit nehmen, die Datei README im Verzeichnis \TPW\DOC durchzulesen, bevor Sie irgendetwas anderes mit Turbo Debugger machen. Diese Datei enthält Informationen, die in allerletzter Minute zusammengestellt wurden und in diesen Handbuch nicht enthalten sind. Außerdem sind alle Dateien beschrieben, die auf den Originaldisketten ausgeliefert wurden.

Sie können den Editor von Turbo Pascal für Windows verwenden, um die README-Datei zu lesen.

Die Datei HELPME!.TDW

Nach der Installation finden Sie im Verzeichnis \TPW\DOC\ außerdem eine Datei mit der Bezeichnung HELPME!.TDW, in der Antworten auf Fragen stehen, die von Anwendern häufig gestellt werden. Sehen Sie diese Datei durch, wenn Sie auf Schwierigkeiten stoßen. Unter anderem werden in der Datei HELPME!.TDW folgende Themen behandelt:

- Syntaktische Unterschiede zwischen TDW und Turbo Pascal für Windows.
- Debuggen von Programmen, zu deren Erstellung mehrere verschiedene Programmiersprachen verwendet wurden.
- Allgemeine Fragen beim Einsatz von Turbo Debugger für Windows.

Die Datei MANUAL.TDW

In dem oben schon erwähnten Verzeichnis \TPW\DOC\ ist auch die Datei MANUAL.TDW gespeichert, die eventuelle Korrekturen und Änderungen des vorliegenden Handbuchs enthält, darüber hinaus aber auch eine Beschreibung der Dienstprogramme TDSTRIP und TDUMP.

Dienstprogramme für TDW

Dieses Programmpaket umfaßt auch einige Dienstprogramme, die über die DOS-Kommandozeile aufgerufen werden. Eines dieser Programme, TDWINST,

beschreibt der Anhang C »Die Installation von TDW«. Die Beschreibung der Programme TDSTRIP und TDUMP finden Sie in der Datei MANUAL.TDW.

Hier nur ein kurzer Hinweis auf die Funktion dieser Dienstprogramme:

TDWINST	erlaubt Ihnen, einige Standardeinstellungen von TDW Ihren Bedürfnissen gemäß zu ändern. Dazu zählen die verwendeten Bildschirmparameter und die Einstellung der Farben.
TDSTRIP	Programm zum Entfernen und Auslagern der Symboltabelle aus der EXE.Datei.
TDUMP	Hilfsprogramm zum Disassemblieren von Modulen.

Hinweis: Um eine Liste der verfügbaren Kommandozeilenoptionen zu den Programmen TDSTRIP.EXE oder TDUMP.EXE zu erhalten, geben Sie nur den Programmnamen ein und drücken ↵. Für die Kommandozeilenoptionen von TDUMP.EXE würden Sie zum Beispiel folgenden Befehl eingeben:

```
TDUMP
```

Die Liste der verfügbaren Kommandozeilenoptionen zu TDWINST erhalten Sie, wenn Sie den Programmnamen gefolgt von der Option -? oder -h eingeben und ↵ drücken, zum Beispiel:

```
TDWINST -?
```

Wie geht's weiter?

Sie können nun beginnen, sich mit Turbo Debugger für Windows vertraut zu machen. Da dieses Handbuch für drei verschiedene Benutzergruppen geschrieben ist, werden ganz verschiedene Kapitel für Sie von besonderem Interesse sein. Die folgende Kurzübersicht wird Ihnen bei der Orientierung helfen.

Neulinge im Umgang mit Turbo Pascal

Wenn Sie Anfänger sind, werden Sie erst einmal lernen wollen, kleine Programme mit Turbo Pascal zu schreiben, bevor Sie den Umgang mit dem Debugger erlernen. Es gibt natürlich keinen besseren Weg, sich Wissen über den Debugger anzueignen, als einen tatsächlichen Fehler aufzuspüren, mit dem Sie in Ihrer eigenen Arbeit konfrontiert sind. Am besten gehen Sie wie folgt vor: Nachdem Sie sich ein Grundwissen in Ihrer Sprache angeeignet haben, arbeiten Sie Kapitel 3, »Einführung in das Debuggen«, durch, eine Einführung in die wichtigsten Funktionen von Turbo Debugger. Sie lernen dort genügend über die Möglichkeiten zum Debuggen Ihres ersten Programms. Die weiterführenden Möglichkeiten des Debuggers werden dann in einem späteren Kapitel behandelt.

Fortgeschritten im Umgang mit Turbo Pascal, aber Neuling mit TDW

Wenn Sie bereits Erfahrung im Programmieren mit Turbo Pascal haben, können Sie sich in Kapitel 2, »Fehlersuche mit Turbo Debugger«, über die interessanten, neuen Möglichkeiten informieren, die Ihnen Turbo Debugger bietet. Sie können dann, je nach persönlichem Arbeitsstil, entweder das Tutorial durcharbeiten, oder gleich zu Kapitel 4, »Der Aufruf von Turbo Debugger«, übergehen. Eine Gesamtübersicht zu allen Befehlen finden Sie in Kapitel 14, »Die Befehlsübersicht«.

Fortgeschrittene im Umgang mit Turbo Debugger

Wenn Sie schon mit der DOS-Version von Turbo Debugger gearbeitet haben, dann sind Ihnen die Grundfunktionen dieses Programms bereits vertraut. Sie können sofort mit Kapitel 11 »Debuggen unter Windows« beginnen. Dieses Kapitel geht auf die Möglichkeiten (und Probleme) des Debuggens unter Windows ein. Wenn Sie in Ihre Programme die *ObjectWindows*-Bibliothek einbinden sind für Sie auch die Kapitel 16 »Ein ObjectWindows-Programm debuggen« und 17 »Debuggen von Windows-Anwendungen« interessant.

2. Fehlersuche mit Turbo Debugger

Wir alle machen Fehler. Ob wir nun einfache Dinge tun, wie spazierengehen, oder komplizierte Dinge, wie programmieren, wir stolpern hin und wieder.

Beim Programmieren kann Stolpern zur Lebensart werden. Sehr wenige Programmierer sind jemals fähig, auf Anhieb ein fehlerfreies Programm zu schreiben. Darüber braucht man sich weder zu schämen noch zu wundern. Aber wer stolpert, muß sich auch wieder aufrappeln, es noch einmal versuchen und dies, so oft man stolpert. Im Jargon des Programmierers nennt man das Debuggen.

Was ist Debuggen?

Debuggen ist das Auffinden und Korrigieren von Fehlern (»bugs«) in Programmen. Es ist nichts außergewöhnliches, mehr Zeit auf die Fehlersuche in einem Programm zu verwenden, als für die gesamte übrige Programmerstellung. Debuggen ist keine exakte Wissenschaft. Das beste Mittel zum Debuggen ist häufig immer noch der eigene Grips. Nichtsdestoweniger kann man sich die Aufgabe durch eine systematische Vorgehensweise erleichtern.

Der Prozeß der Fehlersuche gliedert sich im großen und ganzen in vier Schritte.

1. Einen Fehler erkennen.
2. Den Fehler lokalisieren.
3. Die Fehlerursache finden.
4. Den Fehler korrigieren.

Liegt überhaupt ein Fehler vor?

Der erste Punkt kann sehr offensichtlich sein. Der Computer hängt sich auf, sobald Sie das Programm laufen lassen. Oder er stürzt in einem Schwall von sinnlosen Zeichen ab. Manchmal jedoch sind die Probleme nicht so leicht zu erkennen. Das Programm kann wunderbar laufen, bis eine bestimmte Zahl eingegeben wird, etwa 0 oder eine negative Zahl, oder bis Sie sich das Ergebnis genau ansehen. Möglicherweise bemerken Sie erst dann, daß das Ergebnis vielleicht um einen Faktor von 0,2 nicht stimmt, oder daß die Initialen in einer Namensliste falsch sind.

Wo ist der Fehler?

Der zweite Schritt ist manchmal der schwierigste: den Fehler zu lokalisieren. Sehen wir den Tatsachen ins Auge, man kann einfach kein ganzes Programm auf einmal im Kopf behalten (außer es handelt sich um ein außergewöhnlich kleines). Deshalb ist es am Besten, wenn Sie das Programm in Module aufteilen und diese einzeln debuggen. Die strukturierte Programmierung ist für diese Art der Fehlersuche ideal.

Um was für eine Art von Fehler handelt es sich?

Der dritte Schritt, das Ermitteln der Fehlerursache, ist wohl der zweitschwierigste von allen. Wenn man erst einmal herausgefunden hat, wo der Fehler ist, dann ist es im allgemeinen etwas einfacher zu ermitteln, warum das Programm Zicken macht. Wenn Sie zum Beispiel herausgefunden haben, daß der Fehler in einer Prozedur Ausgabe_der_Namen liegt, müssen Sie nur noch die Zeilen dieser Prozedur untersuchen und nicht mehr das ganze Programm. Nichtsdestoweniger kann der Fehler noch schwer zu ermitteln sein und die Suche nach der Ursache noch einiges Experimentieren erfordern.

Wie kann man den Fehler beheben?

Der letzte Schritt ist schließlich die Korrektur. Bewaffnet mit Ihren Kenntnissen der Programmiersprache und dem Wissen, wo der Fehler liegt, wetzen Sie diesen aus. Nun starten Sie Ihr Programm nochmal, warten auf den nächsten Fehler und fangen wieder von vorne an.

Häufig wird dieser vierstufige Prozess beim Erstellen eines Programms von selbst durchgeführt. Es gibt zum Beispiel eine Menge Syntaxfehler, die das Compilieren eines Programms verhindern, wenn man sie nicht korrigiert. Turbo

Pascal für Windows überprüft während der Compilierung die Syntax und macht Sie auf eventuelle Fehler aufmerksam, die Sie dann sofort entfernen können. (In Kapitel 15, »Ein Standard-Pascal-Programm debuggen«, finden Sie eine ausführlichere Behandlung des Ablaufs der Fehlersuche.)

Andere Fehler sind allerdings um einiges heimtückischer und schwieriger zu erkennen. Sie lauern so lange in ihrem Versteck, bis beispielsweise eine negative Zahl eingegeben wird, oder sie sind so schwer zu fassen, daß man einfach aufgeschmissen ist. Hier tritt Turbo Debugger auf den Plan.

Wozu Sie Turbo Debugger einsetzen können

Wenn Sie Turbo Debugger als alleinstehendes Programm verwenden, haben Sie einen Debugger, der wahrscheinlich sehr viel leistungsfähiger ist als es ein in der Entwicklungsumgebung eingebauter Debugger je sein könnte.

Turbo Debugger kann für jedes Programm verwendet werden, das mit Turbo Pascal für Windows erstellt wurde. Er läuft im Textmodus und erlaubt Ihnen, zwischen Ihrem Programm und dem Debugger hin und herzuschalten.

Sie können Turbo Debugger als Hilfsmittel während der beiden schwierigsten Schritte des Korrekturablaufs verwenden – der Fehlersuche und dem Auffinden der Fehlerursache.

Turbo Debugger hilft Ihnen mit seinen umfangreichen Möglichkeiten, diese Hürden zu überwinden, die Programmausführung zu verlangsamen und das Programm an jeder beliebigen Stelle zu überprüfen. Man kann sogar neue Werte von Variablen testen, um zu prüfen, wie diese das Programm beeinflussen. Diese Möglichkeiten bedeuten in der Praxis, daß man nach Herzenslust *tracen*, den Code *Schritt für Schritt abarbeiten*, *Daten anschauen* sowie *Variablen untersuchen* und *ändern* kann.

Spur verfolgen (Trace)	Das Programm wird Zeile für Zeile abgearbeitet.
Schritte (Step)	Wenn Sie schrittweise einzelne Befehle ausführen, können alle Prozedur- oder Funktionsaufrufe übersprungen werden. Falls Sie sicher sind, daß Ihre Prozeduren und Funktionen fehlerfrei laufen, kann die Fehlersuche auf diese Art beschleunigt werden.
Darstellen (View)	Sie können mit Turbo Debugger ein spezielles Fenster öffnen, in denen Sie ein Dutzend verschiedener Dinge darstellen können: Variablen, deren Werte, Breakpoints, den Inhalt des Stacks, ein Log-Protokoll, eine Datei mit Daten, eine Quelldatei, den CPU-Code, Speicherbereiche, Register, den Status des Arithmetikprozessors oder Programmausgaben.
Untersuchen (Inspect)	Sie können mit Turbo Debugger auch die tieferen Zusammenhänge zwischen den Datenstrukturen in Ihrem Pro-

	gramm nachvollziehen, indem Sie den Inhalt von Array-Elementen und Strukturkomponenten mit Hilfe von Inspector-Fenstern ermitteln.
Verändern (Change)	Man kann den Wert einer Variablen sowohl global als auch lokal durch einen bestimmtem Wert ersetzen.
Beobachten (Watch)	Sie können Programmvariablen herausstellen und über ihre sich ändernden Werte auf dem laufenden bleiben, während das Programm weiterläuft.

Mit derart leistungsfähigen Hilfsmitteln kann man ein Programm in kleine Abschnitte aufteilen und dann nachprüfen, ob ein solcher Teilbereich in Ordnung ist, bevor man den nächsten untersucht. Auf diese Weise können Sie sich durch ein Programm arbeiten, unabhängig wie groß oder kompliziert es ist, bis der Fehler gefunden ist. Vielleicht stoßen Sie dabei auf eine Funktion, in der versehentlich einer Variablen ein Wert zugewiesen wird, vielleicht bleibt das Programm auch in einer Endlosschleife hängen, oder es wird in eine chaotische Rekursion gezogen. Wo auch immer das Problem liegt, Turbo Debugger ist Ihnen eine wesentliche Hilfe beim Auffinden und Identifizieren des Fehlers.

Mit Turbo Debugger können Sie auch Programme debuggen, die in objektorientiertem Pascal erstellt wurden. Das Programm kennt sich aus mit Objekten und Klassen, ist in der Lage, die dynamische Bindung virtueller Methoden richtig zu interpretieren, so daß automatisch immer der richtige Code ausgeführt und dargestellt wird.

Was Turbo Debugger nicht kann

Bei all den Möglichkeiten, die Turbo Debugger bietet, könnte man den Eindruck bekommen, der Debugger könne einfach alles. In Wirklichkeit gibt es mindestens drei Dinge, die Turbo Debugger nicht für Sie erledigen kann:

- Turbo Debugger kann Ihr Programm nicht neu compilieren. Dafür brauchen Sie Turbo Pascal für Windows.
- Turbo Debugger erspart Ihnen nicht das Denken. Wenn Sie ein Programm debuggen, ist Nachdenken Ihr wichtigstes Kapital. Turbo Debugger ist ein leistungsfähiges Werkzeug, gedankenlos angewandt, werden Sie sich jedoch kaum Zeit oder Mühe sparen.
- Turbo Debugger läuft unter Windows nicht im Grafik- sondern im Textmodus.

Wie Turbo Debugger vorgeht

Das wirklich Gute ist: Turbo Debugger bietet höchste Leistungsfähigkeit, während er gleichzeitig leicht, beinahe instinktiv zu bedienen ist. Gekrönt wird diese kunstvolle Mischung aus Leistung und Einfachheit durch eine exzellente Benutzerschnittstelle, über deren Vorteile Sie im nächsten Abschnitt informiert werden.

Die Vorzüge von Turbo Debugger

Haben Sie erst einmal angefangen, mit Turbo Debugger zu arbeiten, werden Sie nicht mehr davon loskommen. Das Programm wurde speziell für leichtes und bequemes Arbeiten entwickelt. Um dies zu gewährleisten, bietet Turbo Debugger folgende Möglichkeiten:

- Bequeme und übersichtliche Menüs.
- Kontextbezogene Popup-Menüs in fast allen Bereichen des Programms. Dadurch wird es praktisch überflüssig, langwierige Befehle im Kopf zu behalten und einzugeben.
- Wenn Sie Text eingeben müssen, führt Turbo Debugger eine Liste von allen Texten mit, die Sie in vergleichbaren Situationen geschrieben haben. Sie können aus diesen Eingaben wählen, den Text editieren oder neuen Text einfügen.
- Volle Makrokontrolle, um Befehlsfolgen und Eingaben zu beschleunigen.
- Bequeme und umfassende Fensterverwaltung.
- Zugang zu verschiedenen Arten von Online-Hilfesystemen.
- Aufzeichnung des Programmablaufs mit der Möglichkeit, das Programm rückwärts ausführen zu können.
- Mausunterstützung.
- Debuggen ist sowohl mit einem als auch mit separaten Bildschirmen möglich.

Menüs und Dialogfenster

Wie viele andere Produkte von Borland, bietet Turbo Debugger ein komfortables Menüsystem mit einer Menüleiste in der obersten Bildschirmzeile. Die Menüleiste ist immer zugänglich, egal, in welchem Fenster Sie sich befinden (zu erkennen am Cursor).

Allen Einträgen auf der Menüleiste sind eigene Untermenüs (*Pulldown*-Menüs) zugeordnet. Mit Hilfe der Pulldown-Menüs können Sie

- einen Befehl aufrufen;
- ein *Popup*-Menü öffnen. Popup-Menüs erscheinen nach der Auswahl von Menüpunkten, hinter den eine dreieckiger Pfeil steht.
- ein *Dialogfenster öffnen*. Dialogfenster erscheinen bei allen Menüpunkten, auf die eine Ellipse (...) folgt.

Bedienung der Menüs

Ins Menüsystem kommen Sie mit einer der folgenden vier Methoden:

- Drücken Sie F10, bewegen Sie dann den Cursor zum gewünschten Menüpunkt. Drücken Sie dann die Eingabetaste ↵.
- Drücken Sie F10 und dann den ersten Buchstaben der Menübezeichnung (*Leertaste*, F, V, R, B, D, O, W , H).
- Drücken Sie die Taste Alt zusammen mit einem der Anfangsbuchstaben der Befehle in der Menüleiste (*Leertaste*, F, V, R, B, D, O, W , H), um das ausgewählte Befehlsmenü zu aktivieren. [Alt][F] bringt Sie zum Beispiel stets ins Menü *File*. Das Menü *System* erhalten Sie mit Alt-*Leertaste*.
- Klicken Sie mit der Maus in den gewünschten Befehl der Menüleiste.

Innerhalb der Menüs können Sie sich wie folgt umherbewegen:

- Drücken Sie die Tasten ← und → , um vom einem Menü ins andere zu gelangen. (Wenn Sie sich zum Beispiel im *File*-Menü befinden, kommen Sie mit →in das *View*-Menü.
- Benützen Sie die Tasten Home und End , um jeweils zum ersten und letzten Menüpunkt zu gelangen.
- Wählen Sie einen Menübefehl aus und drücken Sie ↵, um in ein Untermenü (Popup-Menü) oder Dialogfenster zu gelangen.
- Klicken Sie mit der Maus auf einen Befehl, um ein Untermenü oder Dialogfenster zu aktivieren.

Verlassen wird das Menüsystem durch eine der folgenden Aktionen:

- Drücken Sie Esc, um ein Menü zu verlassen. Solange Sie sich nicht in der obersten Menüebene befinden, kommen Sie dadurch in das Menü, das unmittelbar zuvor aktiviert war.
- Drücken Sie innerhalb eines Pulldown-Menüs Esc und kehren Sie zum aktiven Fenster zurück.
- Mit F10 gelangen Sie aus jeder Menüebene in das unmittelbar zuvor aktivierte Fenster.
- Klicken Sie mit der Maus in das aktive Fenster, um das Menüsystem zu verlassen und zum aktiven Fenster zurückzukehren.

Einige Befehle der Hauptmenüs kann man über spezielle Tastenkombinationen ausführen, die auch als Hotkeys oder Tastenkürzel bezeichnet werden.

Steht ein solcher Hotkey zur Verfügung, so erscheint das entsprechende Tastenkürzel rechts neben dem Menübefehl.

Die Abbildungen in Kapitel 14, »Die Befehlsübersicht«, zeigen den vollständigen Menübaum von Turbo Debugger. In Tabelle 14.1 sind alle Hotkeys aufgelistet. Eine Übersicht über alle Befehle, die Turbo Debugger zur Verfügung stellt, finden Sie ebenfalls in Kapitel 14.

Dialogfenster

Viele Befehlsoptionen von Turbo Debugger werden in *Dialogfenstern* angegeben. Ein Dialogfenster kann aus folgenden Dialogelementen bestehen:

Dialogelement	Beschreibung
Aktionsschalter	Ein Aktionsschalter ist anhand eines unterlegten Texts (auf Monochrom-Bildschirm durch eine inverse Darstellung) gekennzeichnet. Wenn Sie einen Aktionsschalter auswählen, wird die entsprechende Aktion sofort ausgeführt. So läßt sich ein Dialogfenster zum Beispiel durch die Auswahl des Schalters *OK* unter Übernahme der angegebenen Optionen verlassen (das Tastenkürzel für *OK* lautet [Alt][K]), während die Einstellungen mit *Cancel* unter Abbruch des Dialogs verworfen werden können. Dialogfenster enthalten außerdem den Schalter *Help*, mit dem Sie online-Hilfestellungen erhalten.
Markierungsfeld	Ein Markierungsfeld ist ein Ein-/Ausschalter. Damit läßt sich eine Option ein- oder ausschalten. Wenn das Markierungsfeld eingeschaltet ist, erscheint ein *X* in eckigen Klammern: [X].
Schaltfeld	In einem Schaltfeld kann man eine von mehreren Einstellungen vornehmen. Es kann immer nur eine Option ausgewählt werden. Die ausgewählte Option wird durch einen dicken Punkt in runden Klammern angezeigt.
Eingabefeld	In einem Eingabefeld können Strings (zum Beispiel der Name einer Datei) eingegeben werden. Eingabefelder sind häufig mit einer *History-Liste* verknüpft (siehe »Geschichtsstunden«).
Liste	Mit Hilfe der Tasten [Tab] und [Shift][Tab] können Sie sich in Dialogfenstern bewegen. Innerhalb von Auswahlfeldern verwenden Sie die Pfeiltasten.

Wenn Sie mit einer Maus arbeiten, ist die Bedienung eines Dialogfensters noch einfacher. Sie brauchen bloß das gewünschte Dialogelement anzuklicken. Um das Dialogfenster zu schließen, klicken Sie in das Schließsymbol in der linken oberen Ecke.

Hinweis: Sie können die Dialogelemente in einem Dialogfenster auch über deren Hotkeys auswählen. Der Hotkey ist als hervorgehobener Buchstabe des Befehls zu erkennen.

Gewußt wo

Zusätzlich zum praktischen Menüsystem von Borland bietet Turbo Debugger den Vorteil größerer Übersichtlichkeit und verringertem Lernaufwand dadurch, daß sich die Anzahl der Menüs in Grenzen hält.

Um diesen Punkt zu verstehen, müssen Sie sich zunächst klarmachen, daß Turbo Debugger ein Programm ist, das kontextbezogen arbeitet. Turbo Debugger ist stets genau auf dem laufenden darüber, welches Fenster Sie geöffnet haben, welcher Text ausgewählt ist und in welchem Teil des Fensters sich der Cursor befindet. Mit anderen Worten, der Debugger weiß genau, was Sie gerade anschauen und wo der Cursor ist, wenn Sie einen Befehl auswählen. Diese Informationen werden benutzt, wenn der Debugger auf Ihren Befehl reagiert. Folgendes Beispiel erläutert diesen grundlegenden Aspekt. Nehmen wir an, in Ihrem Programm steht folgende Zeile:

```
Zaehler[Grad] := Zaehler[Grad] + 1;
```

Wie Sie bei der Arbeit mit Turbo Debugger feststellen werden, ist es leicht, Informationen über Datenstrukturen zu bekommen. Sie drücken einfach [Ctrl][I], um die Daten zu untersuchen. Wenn der Cursor auf Zaehler steht, informiert Sie Turbo Debugger über den Inhalt der gesamten Array-Variablen. Wenn Sie aber den gesamten Array-Namen zusammen mit dem Index auswählen (d.h. hervorheben) würden und dann [Ctrl][I] drücken, so würde Ihnen Turbo Debugger nur ein Element zeigen.

Auf diese Art können Sie sich in immer feinere Programmdetails vortasten. Wenn Sie ein Array untersuchen, gelangen Sie über [Ctrl][I] zur Ansicht eines bestimmten Elements.

Diese Art der Berücksichtigung des Kontexts gewährleistet eine extrem einfache Bedienung von Turbo Debugger. Sie erspart Ihnen die Mühe, lange Reihen von Menübefehlen oder obskuren Kommandozeilenoptionen im Gedächtnis zu behalten und einzugeben. Sie gehen einfach zu dem Datenobjekt, das Sie untersuchen wollen, oder wählen es mit Hilfe der [Ins]-Taste oder der Maus aus. Dann rufen Sie den Befehl, zum Beispiel [Ctrl][I] für *Inspect*, auf. Turbo Debugger leistet immer sein Bestes, um Sie so gut wie möglich zu bedienen.

Die Kontextbezogenheit, die dem Benutzer das Leben erleichtert, erschwert jedoch die Dokumentation der Befehle. Dies ergibt sich aus der Tatsache, daß zum Beispiel [Ctrl][I] im Turbo Debugger mehrere Funktionen haben kann. Was ein Befehl bewirkt, hängt davon ab, wo der Cursor steht und welcher Text ausgewählt wurde.

Lokale Menüs

Ein anderer Aspekt der Kontextsensibilität liegt in der Verwendung von Popup-Menüs, die der Situation entsprechend erscheinen.

Popup-Menüs in Turbo Debugger nennt man ortsabhängig, weil sie auf den speziellen Punkt zugeschnitten sind, der durch die Position des Cursors bezeichnet ist. Es ist wichtig, (ortsabhängige) Popup-Menüs und allgemeine Menüs auseinanderzuhalten. Hier sehen Sie nun einen Bildschirm, der sich aus beiden

Menüarten zusammensetzt (Sie können einen solchen Bildschirm freilich bei der Arbeit mit Turbo Debugger nicht erzeugen, da beide Menüarten niemals gleichzeitig aktiv sein können):

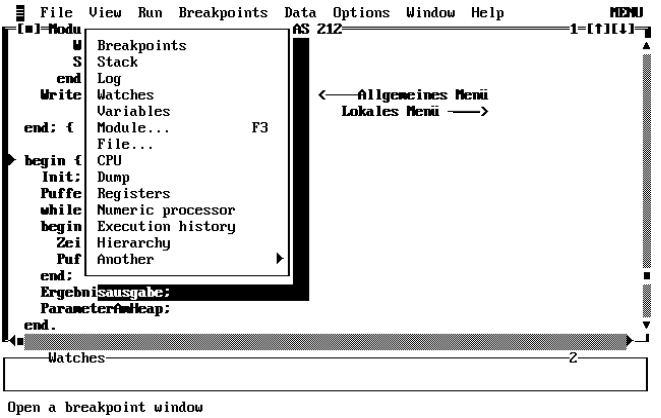


Abbildung 2.1 Menüs der Menüleiste im Vergleich mit Popup-Menüs

Man unterscheidet:

Allgemeine Menüs in der Menüleiste

- Allgemeine Menüs sind diejenigen, zu denen man durch Betätigung der F10-Taste und der Verwendung der Pfeiltasten beziehungsweise durch Eingabe des ersten Buchstaben des Menünamens gelangt.
- Allgemeine Menüs sind immer verfügbar und auf der Menüleiste im oberen Bildschirmteil sichtbar.
- Ihr Inhalt bleibt immer gleich.
- Einige der Menübefehle haben Hotkey-Abkürzungen, die Turbo Debugger in jeder Situation zur Verfügung stellt.

Lokale kontextbezogene Menüs

- Ein Popup-Menü wird aufgerufen durch Eingabe von [Alt][F10] oder [Ctrl][F10]. Am bequemsten geht der Aufruf des lokalen Menüs durch Anklicken und Halten der rechten Maustaste.
- Ort und Inhalt dieses Menüs hängen von der Textauswahl und der Cursorposition ab.
- Die Inhalte von Popup-Menüs können sich ändern. (Auch wenn dies der Fall ist, sollte man sich merken, daß viele der lokalen Befehle in fast allen lokalen Menüs vorhanden sind. Ein bestimmter Grundstock an Befehlen ist also immer vorhanden.) Sogar die Auswirkungen von gleichnamigen Befehlen können je nach Kontext verschieden sein.
- Bei einem Popup-Menü gibt es für jeden Befehl eine Abkürzung, bestehend aus Ctrl und dem ersten Buchstaben des Befehls. Dies bewirkt, daß ein

Hotkey, zum Beispiel [Ctrl][S], in verschiedenen Situationen eine unterschiedliche Bedeutung hat. (Wie oben erwähnt, haben aber alle Popup-Menüs einen Kern von unveränderlichen Grundbefehlen. So führen zum Beispiel der Befehl *Goto* und der Befehl *Search* immer zum gleichen Ergebnis, auch wenn sie von verschiedenen Fensterausschnitten aufgerufen werden.)

Aus der Sicht des Anwenders sind lokale Menüs eine sehr große Erleichterung. Alle möglichen Befehle sind mit einem Blick zu erfassen. Damit wird die Eingabe falscher Befehle verhindert. Außerdem bleiben die Menüs klein und übersichtlich.

Geschichtsstunden

Menüs und kontextbezogene Befehle sind nur zwei Gesichtspunkte der Benutzeroberfläche von Turbo Debugger. Eine weitere Besonderheit, an die man sich schnell gewöhnt, ist die Verfügbarkeit bereits erfolgter Eingaben in einer History-Liste.

Ausgehend von der Philosophie, daß der Benutzer nicht mehr Eingaben als unbedingt nötig machen soll, speichert Turbo Debugger alle Eingaben in den Dialogfenstern und stellt den Text der letzten Eingabe bei einem erneuten Aufruf der Dialogfenster sofort wieder zur Verfügung.

Wenn Sie zum Beispiel die Funktion Investitionsgewinn suchen, dann müßten Sie normalerweise das ganze Wort oder einen Teil des Worts eingeben. Nehmen wir weiter an, daß Sie anschließend eine Variable mit der Bezeichnung MeinAnteil suchen. Wenn diesmal das Dialogfenster erscheint, sehen Sie den Text Investitionsgewinn im Dialogfenster. Wenn Sie nun einen weiteren Text suchen, bleiben die beiden Stringeingaben im Dialogfenster erhalten. Mit weiteren Suchbefehlen verlängert sich die Liste im Dialogfenster.

Das Dialogfenster zum Suchen sieht dann vielleicht wie folgt aus:

Mit Hilfe der History-Liste können Sie die Eingabe für einen Suchvorgang wesentlich beschleunigen. Um die Suche zu starten, brauchen Sie nur noch

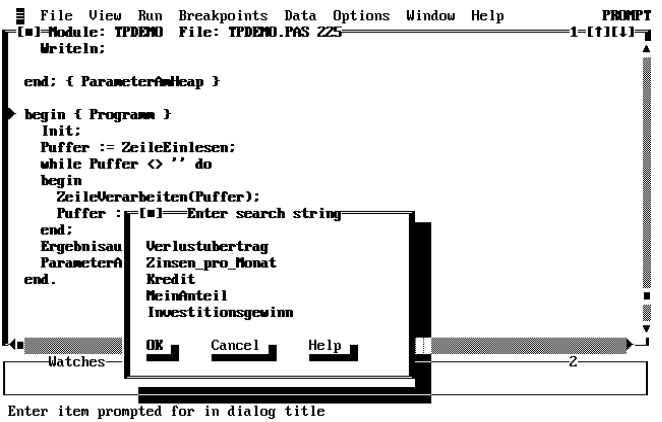


Abbildung 2.2 Eine History-Liste in einem Dialogfenster

einen zuvor eingegebenen Text mit den Pfeiltasten selektieren und dann die Eingabetaste ↵ zu drücken. Wenn Sie einen unveränderten Eintrag aus der History-Liste wiederverwenden, dann finden Sie diesen Eintrag beim erneuten Aufruf der History-Liste an der ersten Stelle. Der Text kann auch editiert werden (man verwendet dazu die Pfeiltasten, um den Cursor im markierten Text umher zu bewegen, und nimmt zum Löschen wie üblich Del oder Backspace). Sie können zum Beispiel MeinAnteil auswählen und in SeinAnteil umwandeln, anstatt den ganzen Text neu einzugeben. Wenn Sie einen neuen Text in einem hervorgehobenen Eintrag eingeben, dann wird der hervorgehobene Eintrag überschrieben. Das erste Objekt in der Suchliste ist immer das Wort, bei dem der Cursor im *Module*-Fenster steht.

Turbo Debugger speichert, falls nicht anders konfiguriert, die letzten fünf Eingaben. (Sie können die Zahl der gespeicherten Einträge mit TDWINST verändern.)

Für die meisten Dialogfenster hält TurboDebugger eine eigene History-Liste bereit. Daher kann es nicht passieren, daß man Suchtexte, wie Zeilennummern oder bestimmte Labels, wegen der vielen Einträge nicht mehr findet.

Automatische Ergänzung des Namens

Es bietet sich immer, wenn Sie nach einem Eintrag für ein Eingabefeld gefragt werden, die Möglichkeit, nur einen Teil der Symbolbezeichnung einzugeben und dann [Ctrl][N] zu drücken.

Achtung: Wenn die Meldung READY... gefolgt von drei Punkten in der rechten oberen Ecke des Bildschirms erscheint, heißt dies, daß die Symboltabelle gerade sortiert wird. [Ctrl][N] funktioniert nicht, solange die drei Punkte am Bildschirm stehen und die Symboltabelle noch in Bearbeitung ist.

- Wenn Sie soviele Anfangsbuchstaben eingegeben haben, daß der Name eindeutig ermittelt werden kann, wird der Namen von Turbo Debugger automatisch ergänzt.
- Wenn der eingegebene Name nicht mit dem Anfang einer Symbolbezeichnung übereinstimmt, passiert gar nichts.
- Wenn durch die Eingabe mehr als ein Symbol gemeint sein könnte, erscheint eine Liste mit den übereinstimmenden Bezeichnungen, aus der Sie die gewünschte auswählen können.

Sukzessive Auswahl

In Turbo Debugger gibt es außerdem die Möglichkeit, die Einträge in einem Dialogfenster nach und nach durch die Eingabe der Anfangsbuchstaben auszuwählen. Wenn der Name in der Liste vorkommt, wird der Auswahlbalken auf die erste Bezeichnung in der Liste gesetzt, die sich mit den Anfangsbuchstaben deckt. Hat man die gewünschte Bezeichnung gefunden, so kann sie mit *OK* ausgewählt werden.

Makros erstellen

Makros sind einfach Tastenkürzel für Tastenfolgen, die Sie frei definieren können. Sie können damit eine Reihe von Turbo-Debugger-Befehlen und Tastatureingaben auf einen einzigen Tastendruck legen und diese somit jederzeit abrufbar machen.

Um ein Makro zu erstellen, drücken Sie die Taste F10, um die Menüleiste zu aktivieren. Anschließend drücken Sie 0, um das Menü *Options* anzuwählen, und rufen dann den Menüpunkt *Macros* auf. Nun stehen Ihnen vier weitere Befehle zur Verfügung: *Create*, *Remove*, *Delete all* und *Stop recording*. Wählen Sie *Create*. Turbo Debugger fordert Sie dann auf, eine Taste anzugeben, auf die das Makro gelegt werden soll. Suchen Sie sich eine selten verwendete oder leicht zu merkende Tastenkombination aus (zum Beispiel [Shift][F1] für »Neustart eines Programms«). Nun geben Sie alle Befehle und Tasten ein, die unter dieser Tastenkombination gespeichert werden sollen.

Um die Aufzeichnung des Makros zu beenden, können Sie eine von drei Methoden verwenden:

- Wählen Sie den Befehl *Options/Macros/Stop recording*.
- Drücken Sie die gerade definierte Makrotaste noch einmal (in unserem Beispiel war das [Shift][F1]).
- Drücken Sie [Alt][-] (halten Sie die Alt gedrückt, bis Sie die Taste mit dem Bindestrich oder dem Minuszeichen gedrückt haben).

Schaufensterbummel

Viele Programme werden heutzutage mit der Fenstertechnik ausgestattet, aber in Turbo Debugger ist das Fensterkonzept ganz besonders ausgeprägt realisiert. Turbo Debugger stellt alle Informationen und Daten in (lokalen und globalen) Menüs, in Dialogfenstern (für Eingaben) und in Fenstern dar. Es gibt viele verschiedene Fenster, je nachdem, welche Art Information dargestellt wird.

Sie öffnen und schließen alle Fenster mit Hilfe von Menübefehlen (oder den Tastenkürzeln für diese Befehle). Die meisten Fenster von Turbo Debugger werden im *View*-Menü geöffnet (dort stehen vierzehn Fenster zur Auswahl). Eine weitere Art von Fenstern wird als *Inspector*-Fenster bezeichnet. Solche Fenster öffnet man mit Hilfe des Befehls *Data/Inspect* oder durch Aufruf des Befehls *Inspect* in einem lokalen Menü.

Fenster des View-Menüs

Es gibt vierzehn Fenster, die im *View*-Menü aufgerufen werden können. Ein geöffnetes Fenster läßt sich mit den Befehlen der Menüs *Window* und *System* verschieben, vergrößern, verkleinern und schließen usw. Diese Menüs werden im Abschnitt »Die Arbeit mit Fenstern« beschrieben.

Module stellt den Programmcode dar, den Sie debuggen. Sie können sich im ganzen Modul frei bewegen und Daten und Code untersuchen, indem Sie auf die Programmvariablen mit dem Cursor zeigen und den entsprechenden Befehl des lokalen Menüs eingeben.

Sie werden wahrscheinlich mehr Zeit im *Module*-Fenster als in allen anderen Fenstern verbringen. Deshalb sollten Sie sich ausreichend Zeit zum Erlernen der verschiedenen lokalen Menübefehle dieses Fensters nehmen.

Sie können ein *Module*-Fenster auch mit [Alt][F3] öffnen. (In Kapitel 8, »Dateien untersuchen und ändern«, werden das *Module*-Fenster und dessen Befehle ausführlich erläutert.)

Watches zeigt Variablen und wie sich deren Werte verändern. Sie können eine Variable in diesem Fenster hinzufügen, indem Sie den Cursor auf eine Variablenbezeichnung setzen und [Ctrl][W] eingeben. In Kapitel 6 »Programmdaten untersuchen und ändern« wird das Fenster *Watches* ausführlich erläutert.

Breakpoints enthält die von Ihnen gesetzten Breakpoints. Ein Breakpoint bezeichnet eine Stelle in Ihrem Programm, an der eine bestimmte Aktion vorgenommen werden soll, zum Beispiel das Anhalten des Programms, damit Sie den Zustand des Programms überprüfen können. Der linke Ausschnitt zeigt die Positionen der einzelnen Breakpoints (oder gibt an, daß der Breakpoint global ist). Im rechten Ausschnitt stehen die Bedingungen, unter denen der gerade ausgewählte Breakpoint ausgelöst wird. Verwenden Sie dieses Fenster, um Breakpoints zu verändern, zu löschen oder hinzuzufügen. (In Kapitel 7, »Breakpoints«, finden Sie eine vollständige Besprechung dieses Fensters und der Funktionsweise der Breakpoints.)

Stack stellt den aktuellen Zustand des Stacks dar, wobei die erste aufgerufene Routine unten steht und alle nachfolgend aufgerufenen Routinen der Reihe nach darüber stehen.

Sie können weitere Informationen über die einzelnen Funktionen oder Prozeduren erhalten, indem Sie den Cursor auf eine solche setzen und diese mit [Ctrl][I] untersuchen (*Inspect*).

Wenn man den Cursor auf eine der Funktionen oder Prozeduren in der Liste bewegt, kann man entweder die lokalen Variablen oder die Argumente, mit denen die Routine aufgerufen wurde, untersuchen. (Kapitel 5, »Steuerung der Programmausführung«, enthält weitere Informationen über das *Stack*-Fenster.)

Log enthält ein Protokoll aller Meldungen. Es speichert eine Liste an Mitteilungen und Informationen, die Sie während der Arbeit mit Turbo Debugger sammeln können (Informationen, warum das Programm angehalten wurde, die Ergebnisse von Breakpoints, die Werte von Strukturen, usw.).

Mit diesem Fenster können Sie auch Informationen über die Speicherbenutzung und Module einer Microsoft Windows-Anwendung erhalten.

Dieses Fenster ermöglicht Ihnen, zurückzublicken und nachzuvollziehen, warum es zu diesem oder jenem Ereignis kommen konnte. Diese Funktion werden Sie sehr schnell zu schätzen wissen, beispielsweise, wenn Sie sich in

einem Programm länger umgesehen haben und sich am Ende fragen, wie Sie wohl hierher gelangt sind. (Kapitel 7, »Breakpoints«, enthält weitere Informationen über das *Log*-Fenster.)

Variables enthält alle Variablen, die momentan von Ihrem Programm aus verwendet werden können. Der obere Ausschnitt enthält die globalen, der untere die lokalen Variablen.

Dieses Fenster ist hilfreich zum Auffinden bestimmter Funktionen oder Variablen, deren Namen man nicht mehr genau weiß. Sie können im globalen Variablenausschnitt nachsehen und so den Namen schnell herausfinden. (In Kapitel 5, »Steuerung der Programmausführung«, wird das Variablen-Fenster ausführlich beschrieben.)

File zeigt den Inhalt einer Datei. Die Datei kann entweder hexadezimal oder als ASCII-Text dargestellt werden. Sie können Strings oder Bytefolgen suchen oder direkt einen bestimmten Teil einer Datei auf einer Diskette patchen.

Diese Möglichkeit empfiehlt sich, wenn Sie ein Programm debuggen, das Dateien verwendet, deren Änderung eine Änderung des Programmverhaltens bewirken sollen. Sie können auch Fehler in Dateien korrigieren oder eine Datei, die vom Programm erzeugt wird, untersuchen, um nachzuprüfen, ob ihr Inhalt korrekt ist. (In Kapitel 8, »Dateien untersuchen und ändern«, erfahren Sie mehr über das *File*-Fenster.)

CPU gibt den aktuellen Zustand der CPU wieder. Dieses Fenster besteht aus fünf Ausschnitten: ein Ausschnitt enthält den disassemblierten Programmcode, ein weiterer zeigt einen Datenbereich hexadezimal an, ein anderer enthält den Stack in Form von Hexadezimalworten, im vierten sehen Sie den Inhalt der CPU-Register und im letzten den Status der CPU-Flags.

Das *CPU*-Fenster werden Sie für die Untersuchung von Maschinensprachebefehlen einer Quellcodezeile oder einzelner Byte einer Datenstruktur benutzen. Wenn Sie den Assemblercode verstehen, können Sie damit auch sehr vertrackte Fehler finden. Für die Mehrzahl der Programme benötigt man dieses Fenster nicht. (In Kapitel 12, »Debuggen auf Assemblerebene«, wird das *CPU*-Fenster näher erläutert.)

CPU-Fenster werden von Turbo Debugger manchmal automatisch geöffnet, wenn das Programm bei einem Befehl angehalten wird, der sich zwischen zwei Quellcodezeilen befindet.

Dump zeigt einen Speicherbereich in der reinen Datendarstellung. Sie können die Daten als Zeichen, Hex-Byte, Worte, Doppelworte und in allen Gleitkommaformaten darstellen. (Dieses Fenster entspricht dem Datenausschnitt des *CPU*-Fensters.) Sie können dieses Fenster verwenden, wenn Sie einen Datenbereich, nicht aber den CPU-Zustand darstellen wollen. Das lokale Menü enthält viele Befehle, mit denen Sie die dargestellten Daten und die Darstellungsart verändern können. Sie können auch ganze Speicherblöcke manipulieren. (In Kapitel 12, »Debuggen auf Assemblerebene«, erhalten Sie weitere Informationen zur Verwendung dieses Fensters.)

Registers enthält die CPU-Register und die Flags. Dieses Fenster hat zwei Ausschnitte, die den Register- und Statusausschnitten des CPU-Fensters entsprechen. Verwenden Sie dieses Fenster, wenn Sie die Register sehen wollen, aber auf den Rest des CPU-Fensters verzichten können. Mit Hilfe der Befehle des lokalen Menüs können Sie die Werte der Register und Flags beliebig manipulieren. (In Kapitel 12, »Debuggen auf Assemblerebene«, erhalten Sie weitere Informationen zur Verwendung dieses Fensters.)

Numeric processor stellt den aktuellen Zustand des Koprozessors dar. Dieses Fenster besteht aus drei Ausschnitten: ein Ausschnitt enthält die Gleitkommaregister, ein weiterer Ausschnitt zeigt die Werte der Statusbits und ein anderer Ausschnitt stellt die Kontrollbits dar.

Dieses Fenster unterstützt die Diagnose von Programmen mit Gleitkommazahlen. Sie müssen ein gutes Verständnis für die Arbeitsweise des Koprozessors mitbringen, wenn Sie von diesem Fenster wirklich profitieren wollen. (In Kapitel 13, »80x87-Koprozessor und Gleitkomma-Emulator«, finden Sie weitere Informationen über das Fenster *Numeric processor*.)

Execution history stellt die Assemblerbefehle und Quellcodezeilen dar, die in Ihrem Programm zuletzt ausgeführt wurden. Der obere Ausschnitt enthält die ausgeführten Assemblerbefehle. Diese Befehle können Sie in umgekehrter Reihenfolge ausführen, und das Programm auf diese Weise »zurücklaufen« lassen. Der untere Ausschnitt zeigt an

1. ob Sie das Programm schrittweise durchgehen;
2. die Nummer der Zeile im Quellcode, die zur Ausführung ansteht;
3. die Zeilennummer im Quellcode.

Sie können diese Fenster zu Ihrer Information heranziehen oder das Programm an eine bestimmte Stelle zurücklaufen lassen. Mehr über dieses Fenster erfahren Sie in Kapitel 5 »Steuerung der Programmausführung«.

Hierarchy enthält eine Liste mit allen Objekten oder Klassen des aktuellen Moduls und stellt deren Beziehungen in einem Hierarchiebaum dar. Das Fenster hat zwei Ausschnitte: einen für die Liste mit den Objekttypen und Klassen sowie einen für den Hierarchiebaum der Objekte.

Dieses Fenster veranschaulicht die Beziehungen zwischen den Objekten und Klassen des aktuellen Moduls. Damit ist es außerdem möglich, einen Objekttyp oder eine Klasse mit allen Ihren Datenfeldern und Methoden oder Datenelementen und Elementfunktionen über die Befehle des lokalen Menüs zu untersuchen. (In Kapitel 10, »Debuggen in objektorientiertem Pascal«, finden Sie weitere Erläuterungen zum *Hierarchy*-Fenster.)

Windows Messages Das Fenster *Windows Messages* zeigt eine Liste der Meldungen, die zwischen den Fenstern Ihrer Windows-Anwendung übertragen wurden. Es besteht aus drei Ausschnitten:

- Im linken Ausschnitt wird angezeigt, für welche Prozeduren oder Handles Sie die Meldungen aufzeichnen.
- Der rechte Ausschnitt zeigt den Meldungstyp.
- Im unteren Ausschnitt werden die Meldungen dargestellt.

Mehrfache Fenster lassen sich für drei Arten von Fenstern öffnen – *Dump*, *File* und *Module* – indem man den Befehl *View/Another* auswählt. Damit können Sie mehrere getrennte Bereiche des Assemblercodes, unterschiedliche Dateien und mehrere Programmbereiche auf einmal betrachten.

Sie können unbesorgt sein, wenn Turbo Debugger eines dieser Fenster von selbst öffnet. Dies passiert als Reaktion auf manche Befehle.

Benutzerbildschirm

Der Ausgabebildschirm ist ein Fenster, das den gesamten Bildschirm mit den Ausgaben Ihres Programms zeigt. Der Bildschirm ist exakt derselbe wie der, den Sie erhalten, wenn Sie Ihr Programm von DOS und nicht von Turbo Debugger aus starten.

Dieses Fenster ermöglicht Ihnen nachzuprüfen, ob Ihr Programm auch tatsächlich da ist, wo Sie es erwartet haben, und ob alle Bildschirmausgaben auch wie erwartet erfolgen. Das Tastenkürzel für dieses Fenster ist [Alt][F5]. Mit dem nächsten Tastendruck kommen Sie wieder zu Turbo Debugger zurück.

Inspector-Fenster

Ein *Inspector*-Fenster stellt den aktuellen Inhalt der ausgewählten Variablen dar. Das Fenster wird normalerweise mit Esc oder durch einen Mausklick auf das Schließsymbol geschlossen. Wenn Sie mehr als ein *Inspector*-Fenster aufgemacht haben, was bei der Untersuchung komplexer Datenstrukturen häufig vorkommen kann, dann können Sie alle Fenster mit F3 oder *Window/Close* auf einmal schließen.

Sie können ein *Inspector*-Fenster öffnen, um sich ein Array von Daten oder den Inhalt einer Variablen oder einen Ausdruck anzuschauen.

Ein *Inspector*-Fenster paßt sich immer an den Datentyp der dargestellten Objekte an. Man kann damit einfache Skalare (int, float usw.) darstellen, aber auch Zeiger, Records, Strukturen und Varianten. Jeder Datentyp wird so dargestellt, daß die Darstellung der Daten so nah wie möglich an die Darstellung in Ihrer Quelldatei herankommt.

Beachten Sie bitte, daß Sie im Gegensatz zu den Fenstern des *View*-Menüs beliebig viele *Inspector*-Fenster mit Hilfe des Befehls *Inspect* erzeugen können. Weitere *Module*-, *File*- und *CPU*-Fenster kann man nur mit dem Befehl *View/Another* erzeugen.

Das aktive Fenster

Auch wenn Sie in Turbo Debugger viele Fenster öffnen können, kann es nur *ein* aktives Fenster geben. Sie können das aktive Fenster anhand der folgenden Kriterien erkennen:

- Das aktive Fenster ist mit einer doppelten, nicht mit einer einfachen Linie eingerahmt.
- Der Titel des aktiven Fensters ist hervorgehoben.
- Der Cursor oder Auswahlbalken befindet sich immer im aktiven Fenster.
- Überlappen sich mehrere Fenster, so ist das oberste Fenster das aktive.

Wenn Sie Befehle auswählen, Texte eingeben, oder Informationen auf und ab rollen, ist ausschließlich das aktive Fenster betroffen. Alle anderen eventuell geöffneten Fenster bleiben unverändert.

Woraus besteht ein Fenster?

Ein Fenster zeichnet sich durch die meisten folgenden Eigenschaften aus, durch die Sie wissen, was Sie mit diesem Fenster machen können:

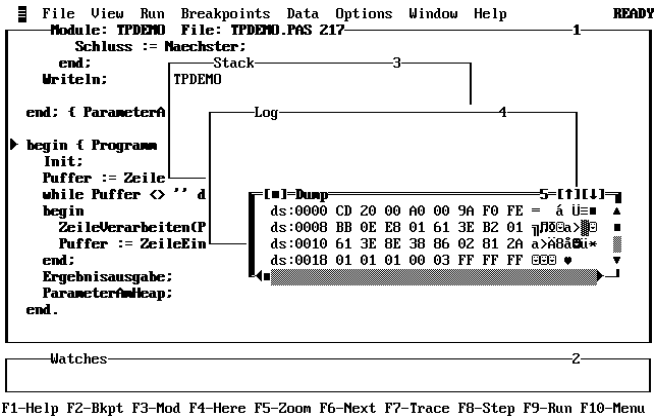


Abbildung 2.3 Können Sie das aktive Fenster erkennen?

- ein Rahmen (doppelt, wenn das Fenster aktiv, sonst einfach);
- einen Titel links oben;
- einer Bildlaufleiste rechts oder unten, wenn das Fenster weitere Informationen enthält, die nicht auf einmal dargestellt werden können. Die Bildlaufleisten können mit der Maus wie folgt bedient werden:
  - ∩ Klicken Sie auf die Richtungspfeile an den beiden Enden der Balken, um das Fenster eine Zeile oder ein Zeichen in die gewünschte Richtung zu verschieben.
  - ∩ Klicken Sie in den grau unterlegten Bereich in der Mitte des Balkens, um den Inhalt um eine Bildschirmseite in die gewünschte Richtung zu verschieben.
  - ∩ Ziehen Sie das Rollsymbol an die gewünschte Position.
- ein Größenänderungsfeld in der unteren rechten Ecke. Ziehen Sie dieses Symbol mit der Maus an die gewünschte Position, um das Fenster zu vergrößern oder zu verkleinern. Wenn sich unten oder rechts an einem Fenster keine Bildlaufleisten befinden, kann die Größe des Fensters auch durch Verschieben dieser Seitenlinien geändert werden.

- eine Fensternummer in der rechten oberen Ecke, mit der die Reihenfolge angegeben wird, in der das Fenster geöffnet wurde.
- ein Zoom- und ein Symbolfeld in der rechten oberen Ecke. Das linke Symbol ist das Zoom-Symbol, das rechte ist das Icon-Symbol. Klicken Sie mit der Maus in diese Symbole, um das Fenster auf volle Bildschirmgröße oder auf die ursprüngliche Größe zu bringen oder in ein Symbol umzuwandeln. (Wenn ein Fenster auf volle Bildschirmgröße ausgedehnt wird, ist nur noch das Symbolfeld verfügbar. Nach der Umwandlung in ein Symbol ist nur noch das Zoomfeld vorhanden.)
- ein Schließfeld in der linken oberen Ecke. Klicken Sie das Feld mit Ihrer Maus an, um das Fenster zu schließen.

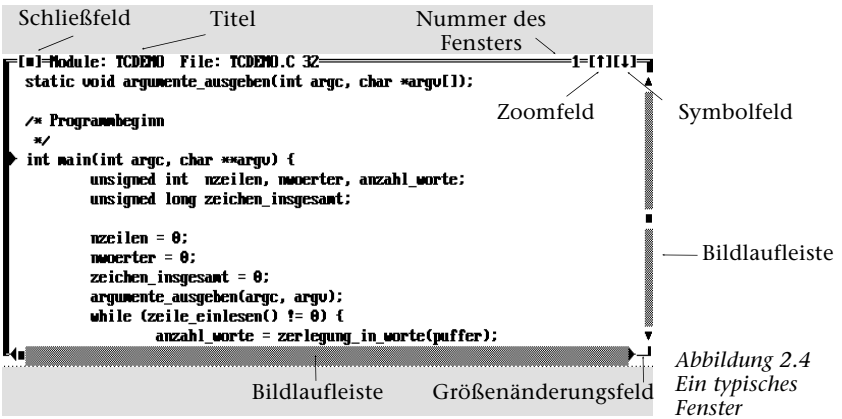


Abbildung 2.4 Ein typisches Fenster

Die Arbeit mit Fenstern

Angesichts der vielen verschiedenen Fenster werden Sie wahrscheinlich häufig mehrere Fenster gleichzeitig geöffnet haben. In Turbo Debugger ist es nicht schwer, von einem Fenster in ein anderes zu springen, ein Fenster umherzubewegen, über ein anderes zu legen, zu verkleinern, falls es Sie bei der Arbeit stört, oder zu vergrößern, damit Sie besser damit arbeiten können, und zu schließen, falls die Bearbeitung abgeschlossen ist.

Die meisten Befehle zur Verwaltung der Fenster befinden sich im Menü *Windows*. Einige weitere Befehle stehen im System-Menü. Dieses Menü erkennen Sie an den drei waagerechten Linien auf der linken Seite der Menüleiste. Mit Alt-Leertaste können Sie das System-Menü öffnen, das Menü *Windows* läßt sich mit [Alt][W] aktivieren.

Von Fenster zu Fenster wechseln Jedes Fenster ist in der rechten oberen Ecke mit einer Numerierung versehen. Normalerweise ist das *Module*-Fenster das Fenster 1 und das *Watches*-Fenster das Fenster 2. Was für ein Fenster Sie anschließend auch öffnen, spielt keine Rolle, es ist in jedem Fall das Fenster 3 usw.

Diese Numerierung ist die Grundlage für eine schnelle Methode, um von einem Fenster in ein anderes zu wechseln. Jedes der ersten neun geöffneten Fenster läßt sich aktivieren, indem man Alt zusammen mit der Nummer des

Fensters, die in der rechten oberen Ecke steht, eingibt. Wenn Sie zum Beispiel [Alt][2] eingeben, wird das *Watches*-Fenster aktiviert. Alle nachfolgenden Befehle beziehen sich nun nur auf dieses Fenster und die darin enthaltenen Einträge.

Mit einer Maus kann ein Fenster einfach durch Anklicken aktiviert werden.

Über das *Window*-Menü erhalten Sie eine Liste der geöffneten Fenster, aus der Sie ein Fenster auswählen können. Mit der Eingabetaste ↵ aktivieren Sie das jeweilige Fenster. (Sie können auch F6 betätigen, um von einem Fenster zum anderen zu wechseln.)

Wenn mehr als neun Fenster geöffnet sind, können Sie trotzdem ein Fenster mit *Window Pick* auswählen; wählen Sie diesen Befehl aus, um sich das gewünschte Fenster in einer Popup-Liste auszusuchen.

Wenn ein Fenster aus einzelnen Ausschnitten – das sind Bereiche in einem Fenster, die für bestimmte Daten verwendet werden – besteht, können Sie mit Tab oder [Shift][Tab] von einem Ausschnitt zum anderen wechseln. Sie können aber auch den Befehl *Window/Next pane* ausführen. Das *CPU*-Fenster ist in Turbo Debugger mit fünf Ausschnitten das Fenster mit den meisten Ausschnitten.

Sie können den gewünschten Ausschnitt auch einfach mit der Maus anklicken.

Wenn Sie von Ausschnitt zu Ausschnitt wechseln, sehen Sie vielleicht manchmal einen blinkenden Cursor, während in manchen Ausschnitten ein Auswahlbalken erscheint. Erscheint ein Cursor, können Sie sich mit den Cursortasten im Text umherbewegen. (Mit PgUp , [Ctrl][Home] und [Ctrl][PgUp] bewegen Sie sich zum Beispiel entweder eine Bildschirmseite nach oben oder in die oberste Zeile des Fensterausschnitts beziehungsweise an die oberste Position in der Liste.) Sie können auch die Wordstar-Befehle für die Cursorbewegungen in einem Fensterausschnitt verwenden. In Kapitel 14, »Die Befehlsübersicht«, finden Sie tabellarische Zusammenfassungen der Tastaturbefehle für die einzelnen Fensterausschnitte.

Erscheint in einem Fenster anstelle eines Cursors ein Auswahlbalken, kann man ebenfalls die Cursortasten verwenden, um einen Eintrag auszuwählen. Es gibt jedoch auch hier eine Reihe von speziellen Tastaturbefehlen. In alphabetisch sortierten Listen kann man zum Beispiel direkt über Tastatureingabe auswählen. Bei jedem eingegebenen Buchstaben springt der Auswahlbalken auf das erste Objekt, das mit dem soeben eingegebenen Buchstaben anfängt. Die Cursorposition in der hervorgehobenen Zeile deutet an, wieviele Zeichen eines Namens bereits eingegeben wurden. Wenn sich der Auswahlbalken über dem gewünschten Objekt befindet, ist die Suche abgeschlossen. Die Suche, die Schritt für Schritt vor sich geht, verringert die Anzahl der Zeichen, die Sie sonst für die Auswahl eines Objekts in einer Liste eingeben müßten.

Sobald ein Objekt in einer Liste ausgewählt ist, können Sie mit [Alt][F10] oder [Ctrl][F10] einen lokalen Befehl aufrufen. In vielen Listen braucht man auch nur die Eingabetaste ↵ zu betätigen, wenn das Objekt selektiert ist. Das ist meist die Kurzform für einen der häufig verwendeten Befehle eines lokalen Menüs. Die jeweilige Funktion der Eingabetaste ↵ in den einzelnen Fensterausschnitten ist im Referenzabschnitt in Kapitel 14, »Die Befehlsübersicht«, beschrieben.

Viele Fensterausschnitte ermöglichen die Eingabe neuer Werte oder Strings, ohne daß vorher ein Befehl ausgewählt wurde. Diese Möglichkeit besteht nur

für den am häufigsten verwendeten Befehl in einem Fensterausschnitt oder einem Fenster (zum Beispiel für *Goto* in einem *Module*-Fenster, für *Search* in einem *File*-Fenster oder für *Change* in einem *Registers*-Fenster).

Fenster anpassen und speichern Wenn Turbo Debugger ein neues Fenster öffnet, erscheint das Fenster in der Nähe der aktuellen Cursorposition. Es erhält eine voreingestellte Größe, die den Anforderungen des Fensters entspricht. Mit Hilfe des Befehls *Window/Size/Move* kann man die Größe und die Position des Fensters verändern. Nach Auswahl dieses Befehls wird der Rahmen des aktiven Fensters mit einer einfachen Linie dargestellt. Sie können dann die Pfeiltasten verwenden, um die Position und die Größe des Fensters auf Ihrem Bildschirm festzulegen. Sind Sie mit der Einstellung zufrieden, drücken Sie die Eingabetaste ↵. (Sie können auch das Tastenkürzel [Ctrl][F5] anstelle von *Window/Size* verwenden.)

Mit einer Maus ist das Verschieben und die Größenveränderung eines Fensters besonders einfach:

- Ziehen Sie das Vergrößerungssymbol in der rechten untern Ecke an die gewünschte Position.
- Ziehen Sie an der Überschrift des Fensters oder den Kanten des Fensters (aber nicht die Bildlaufleisten), um das Fenster umherzubewegen.

Falls Sie ein Fenster kurzfristig vergrößern (oder verkleinern) wollen, können Sie dies mit dem Befehl *Window/Zoom* oder einem Mausklick in das Symbol in der rechten oberen Ecke bewerkstelligen. Das Tastenkürzel für *Window/Zoom* ist F5.

Wenn Sie das Fenster nur vorübergehend entfernen, aber nicht schließen wollen, können Sie das Fenster aktiv machen und anschließend *Windows/Iconize/Restore* auswählen. Das Fenster wird daraufhin in ein kleines Symbol verwandelt, das nur noch den Namen, das Schließfeld und das Zoomfeld enthält. Um das Fenster in die ursprüngliche Form zurückzuverwandeln, aktivieren Sie das Symbol und wählen erneut *Window/Iconize/Restore* aus oder klicken mit der Maus in das Zoomfeld.

Fenster schließen und wiederherstellen Wenn Sie die Arbeit an einem Fenster beendet haben, können Sie das Fenster mit *Window/Undo Close* oder [Alt][F3] schließen.

Wenn Sie mit einer Maus arbeiten, reicht es, in das Schließfeld in der linken oberen Ecke des Fensters zu klicken.

Falls Sie ein Fenster versehentlich schließen, können Sie das Fenster mit *Window/Undo close* oder [Alt][F6] wiederherstellen. Das funktioniert aber nur mit dem zuletzt geschlossenen Fenster.

Sie haben die Möglichkeit, den Bildschirm von Turbo Debugger wieder so herzustellen, als Sie das Programm aufgerufen haben. Wählen Sie dazu den Befehl *System/Restore standard* aus.

Falls Ihr Bildschirm durch die Ausgaben Ihres Programms zerstört wurde (weil Sie vielleicht die Bildschirmzwischenspeicherung abgeschaltet haben), können Sie die Darstellung mit dem Befehl *System/Repaint desktop* wiederherstellen.

Speichern der Fensteranordnung Mit Hilfe des Befehls *Options/*Save kann man eine bestimmte Fenstereinstellung abspeichern. Im Dialogfenster *Save configuration* können Sie das Auswahlfeld *Layout* angeben. Der Bildschirm erscheint beim Aufruf von Turbo Debugger dann immer in dieser Anordnung, falls Sie Ihre Konfiguration in einer Datei mit dem Namen TDCONFIG.TDW abgespeichert haben. Findet Turbo Debugger beim Start diese Konfigurationsdatei, wird sie automatisch geladen. Andere in verschiedenen Dateien gespeicherte Konfigurationen können mit Hilfe des Hauptmenübefehls *Options/Restore options* geladen werden.

Das Hilfesystem

Wie Sie sehen, erleichtert Turbo Debugger die Fehlersuche enorm. Man braucht sich nur sehr wenige Befehle zu merken. Turbo Debugger speichert die vorangegangenen Eingaben, so daß sie später wiederverwendet werden können. Zudem arbeitet man mit einem ausgefeilten Fenstersystem. Turbo Debugger ist ein sehr umfangreiches Programm und stellt zahlreiche Möglichkeiten und Befehle zur Verfügung. Um Unklarheiten zu vermeiden, bietet Turbo Debugger die folgenden Hilfsfunktionen an:

- Eine Statusanzeige in der rechten oberen Ecke zeigt immer den aktuellen Vorgang an. Wenn sich zum Beispiel der Cursor in einem Fenster befindet, dann steht in der Vorgangsanzeige READY (engl. bereit). Ist ein Menü sichtbar, erscheint MENU. Wenn ein Dialogfenster aktiviert ist, steht dort PROMPT. Wenn Sie einmal nicht wissen, was in Turbo Debugger geschieht, brauchen Sie sich nur die Statusanzeige anschauen. (Weitere Möglichkeiten für die Statusanzeige sind SIZE/MOVE, MOVE, ERROR, RECORDING, WAIT, RUNNING, MENU, HELP, STATUS und PLAYBACK.)

- Merken Sie sich, daß das aktive Fenster immer das oberste Fenster ist und von einer doppelten Linie eingerahmt wird.

- Mit F1 können Sie auf ein umfangreiches, kontextbezogenes Hilfesystem zugreifen.

- Die unterste Zeile des Bildschirms liefert Ihnen immer einen schnellen Überblick über die möglichen Tastaturbefehle. Die Zeile hängt ebenfalls vom jeweiligen Kontext ab. Sie ändert sich auch, wenn Sie die Tasten Shift, Alt oder Ctrl drücken.

Zu den beiden letzten Punkten erhalten Sie in den nachfolgenden Abschnitten noch weitere Informationen.

Online-Hilfe

Turbo Debugger enthält, wie alle anderen Borland-Produkte, ein kontextbezogenes Hilfesystem, das mit einem Tastendruck jederzeit online verfügbar ist. Sie erhalten also zu jedem beliebigen Zeitpunkt Hilfestellungen. Ob Sie sich in

einem Menü, einem Fenster oder in einer Dialogfenster befinden oder ob Sie eine Fehlermeldung erhalten, spielt dabei keine Rolle.

Mit F1 erscheint ein Hilfefenster, das Informationen anzeigt, die dem aktuellen Kontext entsprechen. Mit einer Maus kann man auch einfach F1 in der Statuszeile anzeigen. Hilfefenster enthalten hervorgehobene Stichwörter, mit denen man weitere Informationen abrufen kann. Verwenden Sie die Cursortasten, um zum gewünschten Stichwort zu gelangen, und drücken Sie die Eingabetaste ↵, um die Informationen zu diesem Stichwort abzurufen. Mit Hilfe der Tasten Home und End können Sie zum ersten beziehungsweise letzten Stichwort auf dem Bildschirm springen.

Hilfestellungen erhalten Sie auch durch die Auswahl des Menüs *Help* in der Menüleiste ([Alt][H]).

[Alt][F1] dient zur Rückkehr in den vorherigen Hilfsbildschirm des Hilfesystems. Innerhalb des Hilfesystems kann man mit PgUp die letzten 20 Hilfebildschirme zurückgehen. PgDn funktioniert nur in einer Gruppe von verwandten Bildschirmen. Um den Index des Hilfesystems zu erhalten, müssen Sie innerhalb des Hilfesystems die Taste F1 drücken. Mit Esc verlassen Sie das Hilfesystem.

Die Statuszeile

Wo auch immer Sie sich in Turbo Debugger befinden, in der untersten Zeile des Bildschirms erscheint grundsätzlich immer eine kurze Übersicht über die möglichen Tastaturbefehle. Damit haben Sie, dem Kontext entsprechend, immer die wichtigsten Tastaturbefehle auf einen Blick.

Im Fenster Die Statuszeile zeigt alle Befehle an, die mit den Funktionstasten ausgeführt werden können:

```
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

Wenn Sie die Alt-Taste eine oder zwei Sekunden lang gedrückt halten, werden die Befehle angezeigt, die mit der Kombination Alt-Funktionstaste ausgeführt werden können.

```
Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-Instr F8-Rtn F9-To F10-Local
```

Wenn Sie die Ctrl-Taste eine oder zwei Sekunden lang gedrückt halten, werden die Befehle angezeigt, die mit der Kombination Ctrl-Funktionstaste ausgeführt werden können.

```
Ctrl: G-Goto O-Origin F-Follow C-Caller P-Previous S-Search V-View M-Mixed N-New
```

Diese Hilfsmenüzeile ändert sich entsprechend dem aktuellen Fenster und dem aktuellen Fensterausschnitt und zeigt die Kurzformen der Befehle der lokalen Menüs an. Gibt es in einem lokalen Menü mehr Befehle als Erklärungen in einer Zeile Platz haben, werden nur die ersten Befehle dargestellt. Mit [Alt][F10] oder [Ctrl][F10] können Sie alle verfügbaren Befehle eines lokalen Menüs durch Anzeige des gesamten Menüs aktivieren.

Wenn Sie eine Maus haben, brauchen Sie nur in die Befehle der Statuszeile zu klicken, um einen Alt- oder Ctrl-Befehl auszuführen.

In Menüs oder Dialogfenstern Jedesmal, wenn Sie sich in einem Menü oder Dialogfenster befinden, enthält die Statuszeile eine einzeilige Erklärung des aktuellen Menüpunkts. Wenn zum Beispiel *View/Registers* aktiviert ist, steht in der Statuszeile die Meldung `Open a CPU registers window.`

In der Statuszeile erhalten Sie auch für lokale Menübefehle Hilfestellungen.

3. Einführung in das Debuggen

Falls Sie zu den Menschen zählen, die Turbo Debugger möglichst schnell ausprobieren möchten, dann ist dieses Kapitel genau das richtige für Sie. Hier wird Ihnen genügend Wissen darüber vermittelt, wie Sie ein Programm debuggen können, auch wenn Sie noch nicht das ganze Handbuch durchgearbeitet haben. Wenn Sie die hier beschriebenen grundlegenden Konzepte erlernt haben, können Sie das restliche Know-How über die hochintegrierte, intuitive Benutzeroberfläche und das kontextbezogene Hilfesystem durch Ausprobieren erlernen.

In diesem Kapitel werden alle grundlegenden Eigenschaften von Turbo Debugger erklärt. Nach der Beschreibung des Beipielprogramms TDDEMO auf einer der Lieferdisketten erfahren Sie, wie man

- Programme startet und anhält,
- die Inhalte von Programmvariablen ermitteln kann,
- komplexe Datenobjekte, wie Arrays und Strukturen überprüft und
- den Inhalt von Variablen verändert.

Das Beispielprogramm

Anhand des Beispielprogramms TDDEMO.PAS werden die zwei wichtigsten Vorgänge gezeigt, die man kennen muß, um ein Programm zu debuggen: wie man ein Programm unterbricht und wieder startet und wie man Variablen und Datenstrukturen durchsuchen kann. Das Programm selbst ist als Beispiele gedacht. Ein Teil des Codes und der Datenstrukturen wurde nur zur Demonstration der Fähigkeiten von Turbo Debugger eingefügt.

Das Programm erwartet die Eingabe einiger Zeilen Text. Danach werden die eingegebenen Wörter und Zeichen gezählt. Am Ende des Programms erscheint schließlich eine kurze Statistik des eingegebenen Texts, einschließlich der durchschnittlichen Anzahl der Wörter pro Zeile und der Häufigkeit der einzelnen Buchstaben. Das Programm verwendet zur Ausgabe und zur Entgegennanhme von Text die Unit *WinCrt*.

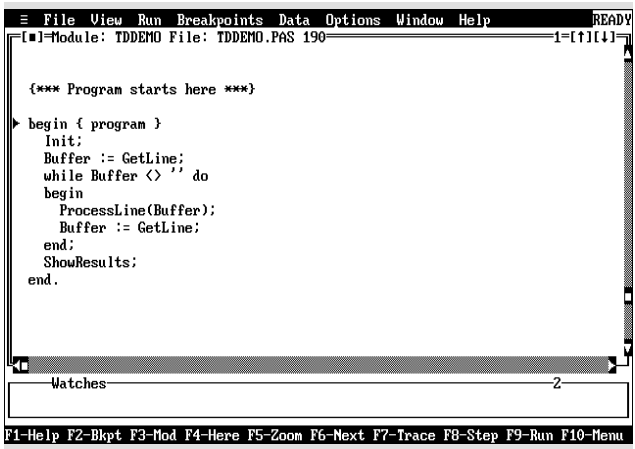


Abbildung 3.1
Der Bildschirm nach dem Start von TDDEMO

Vergewissern Sie sich, daß das Verzeichnis, in dem Sie arbeiten, die beiden Dateien TDDEMO.PAS und TDDEMO.EXE enthält, die Sie für diesen Lehrgang brauchen.

Debugger starten

So müssen Sie vorgehen, wenn Sie das Beispielprogramm debuggen wollen:

1. Vergewissern Sie sich zunächst, daß Windows im Standard- oder erweiterten 386er-Modus arbeitet, TDW läuft nicht im Real-Modus.

2. Aktivieren Sie im Programm-Manager von Windows die Gruppe, die Turbo Pascal enthält, und klicken Sie dessen Symbol zweimal an.

3. Wählen Sie in der Entwicklungsumgebung von Turbo Pascal die Option *File/Open* und laden Sie TDDEMO.PAS.

4. Ist die Datei in das Editorfenster geladen, wählen Sie *Run/Debugger*, um Turbo Debugger zu starten. Wenn notwendig, wird Turbo Pascal das Programm neu kompilieren, damit der Debugger über die Informationen verfügt, die notwendig sind, damit er den Programmablauf kontrollieren kann.

Turbo Debugger lädt darauf das Demoprogramm, zeigt den Startup-Bildschirm und setzt den Cursor an den Anfang des Programms.

Sie sehen auf diesem Bildschirm die Menüleiste, das *Module*-Fenster, das *Watches*-Fenster und die Statuszeile.

Debugger verlassen

Wenn Sie irgendwann den Lehrgang verlassen und auf die DOS-Ebene zurückkehren wollen, dann geben Sie [Alt][X] ein. Falls Sie sich während des Lehrgangs einmal hoffnungslos verirren sollten, können Sie mit [Ctrl][F2] das Programm neu laden und noch einmal beginnen. [Ctrl][F2] löscht jedoch weder Breakpoints noch Angaben im *Watches*-Fenster. Dafür müßten Sie [Alt][F][O] eingeben.

Hilfestellungen

Mit der Taste F1 erhalten Sie jederzeit Hilfeinformationen, sei es über ein Fenster, einen Befehl, ein Dialogfenster oder eine Fehlermeldung. Man lernt eine Menge, wenn man sich Schritt für Schritt durch das Menüsystem bewegt und zu jedem Befehl mit F1 die entsprechende Erklärung abruft.

Turbo Debugger im Einsatz

Der Bildschirm erschient nach dem Start von TDW dreigeteilt, wobei den größten Anteil der Fensterbereich einnimmt, der nach oben durch die Menüleiste und nach unten durch die Statuszeile begrenzt wird.

Die Menüleiste

In der obersten Zeile des Bildschirms befindet sich die Menüleiste. Um ein Menü daraus zu aktivieren, drücken Sie entweder die Taste F10 oder die Taste Alt in Kombination mit dem ersten Buchstaben einer Menübezeichnung.

Drücken Sie jetzt die Taste F10. Sie werden beobachten, daß der Cursor aus dem *Module*-Fenster verschwindet und daß der Befehl *File* in der Menüleiste hervorgehoben wird. Auch die unterste Bildschirmzeile verändert sich, um anzuzeigen, welche Befehle im *File*-Menü zur Verfügung stehen.

Benutzen Sie die Pfeiltasten, um im Menüsystem herumzuwandern. Geben Sie ↓ ein, um das in der Menüleiste hervorgehobene Menü zu aktivieren.

Ein Menü kann man auch durch Anklicken in der Menüleiste mit der Maus öffnen.

Drücken Sie jeweils Esc, um von den verschiedenen Ebenen des Menüsystems zurückzukehren. Ist nur eine Menübezeichnung auf der Menüleiste hervorgehoben, können Sie über Esc ins *Module*-Fenster zurückkehren. Die Menüleiste ist dann nicht mehr aktiviert.

Die Statuszeile

Die unterste Zeile des Bildschirms zeigt wichtige Funktionstasten und deren Auswirkung an.

Die Zeile verändert sich, je nachdem, welche Eingaben (Menübefehle, Eingaben in Dialogfenstern usw.) Sie machen. Halten Sie zum Beispiel die Alt-Taste ein bis zwei Sekunden gedrückt. Sie sehen jetzt, daß sich die unterste Zeile verändert und die Funktionstasten zeigt, die mit Alt kombiniert werden können.

Drücken Sie nun eine Sekunde lang auf die Taste Ctrl. In der untersten Zeile erscheint eine Reihe von Befehlen. Es sind dies die Tastenkürzel der *Befehle des lokalen Menüs* im soeben aktivierten Fensterausschnitt. Diese Befehle ändern sich je nach Art des aktiven Fensters und Fensterausschnitts. Sie werden später mehr darüber erfahren.

Sobald Sie in das Menüsystem übergehen, ändert sich die Statuszeile erneut, um anzuzeigen, was die gerade hervorgehobene Option macht. Mit F10 kommen Sie in die Menüleiste. Drücken Sie danach einmal →, um das Menü *File* auszuwählen. In der Statuszeile steht nun die Meldung »File oriented functions«.

Mit ↓können Sie die Optionen des *File*-Menüs durchgehen und verfolgen, wie sich die Meldung ändert. Drücken Sie Esc oder klicken Sie mit der Maus in das *Module*-Fenster, um das Menüsystem zu verlassen.

Die Fenster

Der Fensterbereich nimmt den größten Teil des Bildschirms ein. Hier untersuchen Sie einzelne Teile Ihres Programms, indem Sie es in verschiedenen Fenstern darstellen.

Zu Beginn erscheinen zwei Fenster auf dem Bildschirm: ein *Module*-Fenster und das *Watches*-Fenster. Solange man keine weiteren Fenster öffnet oder diese beiden Fenster nicht verschiebt, füllen sie zusammen die ganze Bildfläche aus, ohne sich zu überlappen. Neue Fenster überlappen automatisch die alten, bis sie verschoben oder gelöscht werden.

Sie werden feststellen, daß das *Module*-Fenster durch eine doppelte Linie begrenzt und mit einer hervorgehobenen Überschrift versehen ist. Daraus ist ersichtlich, daß eben dieses Fenster gerade aktiviert ist. Mit Hilfe der Cursortasten (Pfeiltasten, Home , End , PgUp , PgDn) können Sie sich innerhalb des aktivierten Fensters bewegen. Mit der Funktionstaste F6 wechselt man in ein anderes Fenster. Tun Sie das jetzt. Das *Watches*-Fenster wird aktiviert und

Abbildung 3.2 Die Fenster Module und Watches ohne Überlappung

erscheint mit einer unterlegten Überschrift und einer doppelten Begrenzungslinie.

Verwenden Sie den Menübefehl *View* aus der Menüleiste, um weitere Fenster zu öffnen. Gehen Sie mit [Alt][V] in das Menü *View* und geben Sie dann [S] ein, um ein *Stack*-Fenster zu öffnen. Das Menü verschwindet während das *Stack*-Fenster über dem *Module*-Fenster erscheint.

Um das aktivierte Fenster zu löschen, drücken Sie bitte [Alt][F3]. Tun Sie das jetzt. Das *Stack*-Fenster verschwindet.

Das zuletzt geschlossene Fenster wird von Turbo Debugger automatisch gespeichert. Sie können es also bei Bedarf jederzeit wieder abrufen. Falls Sie ein Fenster aus Versehen löschen, geben Sie [Alt][W] ein, um ins Menü *Window* zu gelangen. Wählen Sie daraus durch Eingabe von [U] den Befehl *Undo close* und das *Stack*-Fenster erscheint wieder. Sie können auch [Alt][F6] eingeben, um das zuletzt geschlossene Fenster wieder zu restaurieren.

Im Menü *Window* stehen die Befehle, mit denen man die Fenster, die sich auf dem Bildschirm befinden, verändern kann. Man kann sowohl die Fenster auf dem Bildschirm umher bewegen, als auch ihre Größe verändern. (Diese Veränderungen können Sie auch mit Hilfe von [Ctrl][F5] vornehmen.)

Wählen Sie *Window/Size/Move* aus und verwenden Sie die Pfeiltasten, um das aktive Fenster (das *Stack*-Fenster) auf dem Bildschirm neu anzuordnen. Anschließend halten Sie die Taste [Shift]gedrückt. Nun können Sie durch Betätigen der Pfeiltasten die Größe des Fensters verändern. Drücken Sie die Eingabetaste [↵], wenn Sie mit Größe und Position Ihres Fensters zufrieden sind.

Zur Vorbereitung auf den nächsten Abschnitt muß das Stack-Fenster durch Eingabe von [Alt][F3] wieder entfernt werden.

Das Beispielprogramm TDDEMO

Der ausgefüllte Pfeil in der linken Spalte des *Module*-Fensters zeigt an, wo Turbo Debugger Ihr Programm angehalten hat. Da Sie noch nicht mit Ihrem Programm gearbeitet haben, zeigt der Pfeil auf die erste Zeile Ihres Hauptprogramms. Geben Sie [F7] ein, um eine einzelne Quellzeile abzuarbeiten. Nun befinden sich Pfeil und Cursor in der nächsten Zeile.

Beachten Sie die Angabe, die in der Überschrift des *Module*-Fensters rechts steht. Sie bezeichnet die Zeile, in der der Cursor steht. Wenn Sie nun den Cursor mit den Pfeiltasten auf und ab bewegen, können Sie beobachten, wie die Numerierung der jeweiligen Zeile in der Überschrift wechselt.

Damit das Programm bis Zeile 189 ausgeführt wird, bewegen Sie bitte den Cursor in diese Zeile und drücken dann [F4]. TDDEMO fordert Sie zur Eingabe eines Strings auf. Geben sie einige Zeichen ein und drücken Sie dann die Eingabetaste [↵]. Nun – der Cursor ist immer noch in Zeile 189 – geben Sie [F7] ein, um nur eine weitere Zeile des Quellcodes auszuführen. Weil durch diese Zeile ein anderes Unterprogramm aufgerufen wurde, erscheint der Pfeil jetzt in der ersten Zeile der Funktion ZeileVerarbeiten. Drücken Sie nun weiterhin [F7],

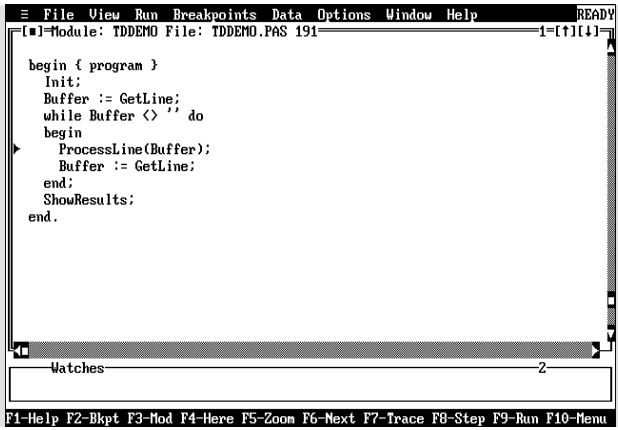


Abbildung 3.3 Das Programm hält sofort nach der Rückkehr von der Funktion argumente_zeigen an

so würden Sie die Funktion ZeileVerarbeiten schrittweise abarbeiten, bis Sie schließlich zur aufrufenden Funktion zurückkehren und dort eine Zeile (192) hinter dem Funktionsaufruf landen würden. Geben Sie stattdessen [Alt][F8] ein, damit das Programm die gesamte Routine durchläuft, aber sofort stoppt, wenn ZeileVerarbeiten beendet wird. Dieser Befehl ist sehr praktisch, wenn Sie eine Routine auf einen Schlag ausführen wollen.

Wenn Sie in Zeile 189 statt F7 die Taste F8 gedrückt hätten, wären Sie direkt ohne Umweg über die Prozedur zu Zeile 192 gekommen. Die Einzelschrittbefehle F8 und F7 sind sehr ähnlich. Der einzige Unterschied besteht darin, daß F8 eine Funktion bei einem Funktionsaufruf überspringt, während F7 den Quellcode der Funktion schrittweise durcharbeitet.

Um ein Programm bis zu einer bestimmten Stelle auszuführen, können Sie auch, anstatt eine Markierung durch den Cursor zu setzen, die Funktion oder die Zeilennummer direkt eingeben. Drücken Sie die Tasten [Alt][F9], um ein Programmsymbol zu bestimmen, das Sie erreichen wollen. Geben Sie ZeileEinlesen ein und drücken Sie die Eingabetaste ↵. Das Programm läuft dann bis zum Beginn der Funktion ZeileEinlesen.

Breakpoints

Die Verwendung von Breakpoints bietet Ihnen eine weitere Möglichkeit zu kontrollieren, wo Ihr Programm anhalten soll. Die einfachste Art, einen Breakpoint zu setzen, ist die Eingabe von F2. Gehen Sie mit dem Cursor in Zeile 141 und betätigen Sie die Funktionstaste F2. Turbo Debugger unterlegt nun die Zeile und zeigt damit an, daß hier ein Breakpoint gesetzt ist. Ein schnellerer Weg, in eine bestimmte Zeile zu gelangen, bietet die Tastenkombination [Ctrl][L]. Dieses Tastenkürzel aktiviert die Option *Line* im laokalen Menü des Fensters *Module*. Sie brauchen dann nur noch die Nummer der gewünschten Zeile im Eingabefeld einzutragen.

Mit der Maus lassen sich Breakpoints besonders einfach ein- und ausschalten. Man braucht bloß in die ersten beiden Spalten des *Module*-Fensters zu klicken.

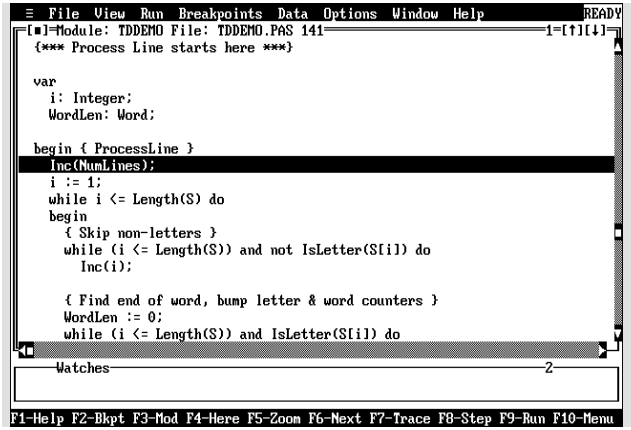


Abbildung 3.4
Ein Breakpoint in Zeile 44

Drücken Sie nun F9, um Ihr Programm ohne Unterbrechung auszuführen. Das Programm erscheint auf dem Bildschirm. Das Demoprogramm läuft jetzt und wartet auf Ihre Texteingabe. Geben Sie abc, Leertaste, und def ein, drücken Sie anschließend die Eingabetaste ↵. Die Bildschirmdarstellung kehrt zu Turbo Debugger zurück. Sie sehen den Pfeil auf Zeile 141, an der Sie zum Abbrechen des Programms einen Breakpoint gesetzt haben.

Eine vollständige Beschreibung der Breakpoints, einschließlich der bedingungsabhängigen und globalen Breakpoints, finden Sie in Kapitel 7, »Breakpoints«.

Das Watches-Fenster

Das *Watches*-Fenster am unteren Bildschirmrand zeigt den Wert von Variablen an, die Sie untersuchen wollen. Um beispielsweise den Wert der Variablen AnzahlWoerter zu bestimmen, bewegen Sie den Cursor auf den

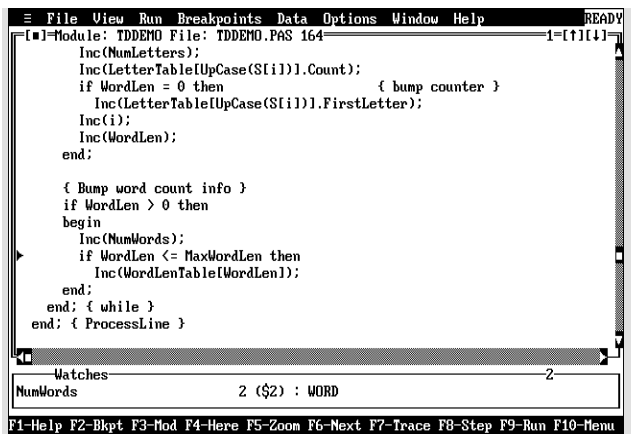


Abbildung 3.5
Eine Variable im Watches-Fenster

Variablennamen in Zeile 164. Geben Sie [Ctrl][W] ein. Dies ist die Abkürzung für den Befehl [Alt][F10][W] im lokalen Menü.

Sie können auch mit der Maus auf [Ctrl][W] in der Statuszeile klicken.

Die Variable `AnzahlWoerter` erscheint jetzt zusammen mit der Angabe des Datentyps (`word`) und dem Inhalt im *Watches*-Fenster. Während Sie Ihr Programm ausführen, bringt Turbo Debugger diesen Wert immer wieder auf den neuesten Stand, um stets den aktuellen Wert der Variablen zu zeigen.

Einfache Datenobjekte in Pascal untersuchen

Haben Sie Ihr Programm einmal auf diese Weise angehalten, bietet sich Ihnen, mit Hilfe des Befehls *Inspect*, eine Anzahl von Möglichkeiten, Daten zu untersuchen. Dieser besonders leistungsfähige Teil des Debugger erlaubt es Ihnen, Daten genau so zu überprüfen, wie Sie sie beim Programmieren vor Augen haben.

Mit den *Inspect*-Befehlen (in den verschiedenen ortsabhängigen Menüs und im Menü *Data*) können Sie jede Variable untersuchen. Angenommen Sie wollen sich den Wert der Variablen `AnzahlZeilen` anschauen. So bewegen Sie den Cursor auf Zeile 141 zurück und setzen ihn auf einen der Buchstaben von `AnzahlZeilen`. Drücken Sie anschließend [Ctrl][I] und ein *Inspector*-Fenster erscheint.

In der ersten Zeile steht der Name der Variablen. Die zweite Zeile zeigt ihre Speicheradresse. In der dritten Zeile sehen Sie den Datentyp von `AnzahlZeilen` (ein word in Pascal) und den aktuellen Wert der Variablen.

Wenn Sie mit der Überprüfung der Variablen fertig sind, geben Sie [Esc] ein, um das *Inspector*-Fenster zu schließen. Genau wie alle anderen Fenster können Sie das *Inspector*-Fenster auch mit [F3] schließen.

Lassen Sie uns diesen Vorgang kurz zusammenfassen. Durch die Eingabe des [Ctrl]-Tastenkürzels gelangten Sie zu den Befehlen des lokalen Menüs im *Module*-Fenster, anschließend über [I] zum Befehl *Inspect*.

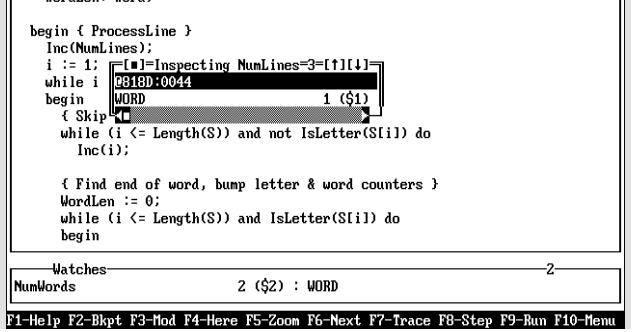


Abbildung 3.6 Ein Inspector-Fenster

Um Daten zu inspizieren, die nicht gleich so günstig im *Module*-Fenster erscheinen, drücken Sie bitte [Alt][D][I]. Damit erhalten Sie ein Dialogfenster auf dem Bildschirm, in dem Sie die Variable eingeben, die Sie untersuchen möchten. Geben Sie nun `BuchstabenTabelle` ein und drücken Sie die Eingabetaste ↵. Es erscheint ein *Inspector*-Fenster, das den Wert von `BuchstabenTabelle` zeigt. Gehen Sie mit Hilfe der Pfeiltasten durch die 26 Elemente, aus denen `BuchstabenTabelle` besteht. Die Überschrift des *Inspector*-Fensters zeigt Namen und Typ der Daten, die sie gerade untersuchen. Im nächsten Abschnitt wird erklärt, wie man solch eine zusammengesetzte Datenstruktur untersucht.

Zusammengesetzte Datenstrukturen in Pascal untersuchen

Ein zusammengesetztes Datenobjekt, wie zum Beispiel ein Array oder eine Struktur, enthält verschiedenartige Komponenten. Gehen Sie auf das vierte Element des Arrays `BuchstabenTabelle` (also auf `['D']`). Geben Sie [Alt][F10] ein, um in das lokale Menü des *Inspector*-Fensters zu gelangen. Drücken Sie dann [I], um den Befehl *Inspect* zu aktivieren. Ein neues *Inspector*-Fenster erscheint und zeigt den Inhalt dieses Array-Elements. Das *Inspector*-Fenster in Abbildung 3.7 zeigt den Inhalt einer Struktur des Typs `ZInfoRec`.

Wenn Sie den Cursor auf einen der Feldnamen positionieren, erscheint der Datentyp dieses Feldes im unteren Fensterausschnitt des *Inspector*-Fensters.

Würde eines dieser Fenster wiederum eine zusammengesetzte Datenstruktur enthalten, könnten Sie wieder einen *Inspect*-Befehl ausführen und noch tiefer in die Datenstruktur einblicken.

Drücken Sie [Alt][F3], um beide *Inspector*-Fenster zu schließen und in das *Module*-Fenster zurückzukehren. (Durch die Eingabe von [Alt][F3] können Sie bequem mehrere *Inspector*-Fenster auf einmal entfernen, während [Esc] immer nur das zuletzt bearbeitete Fenster schließt.)

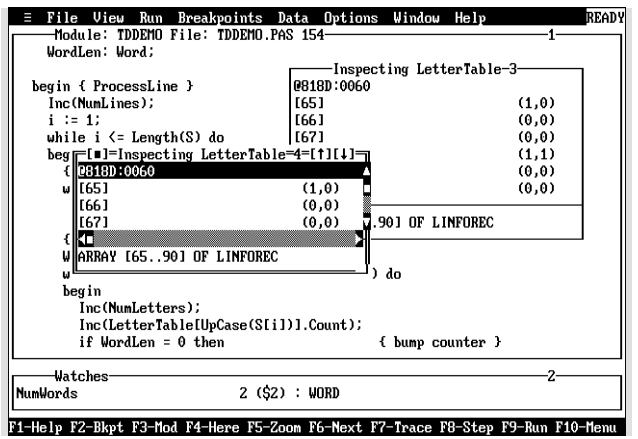


Abbildung 3.7
Die Untersuchung eines Arrays

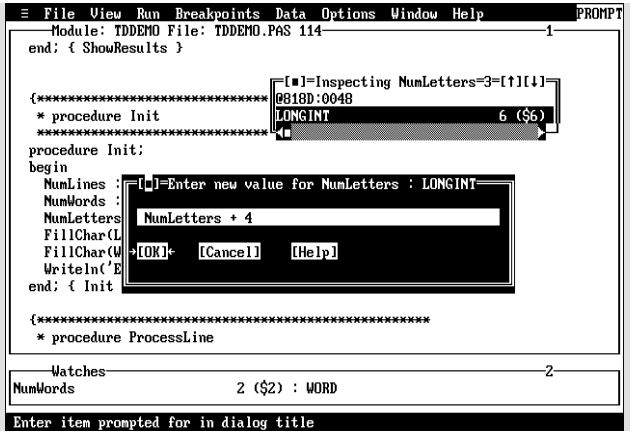


Abbildung 3.8 Das Dialogfenster Change zum Ändern von Daten

Pascal-Daten ändern

Bisher haben Sie gelernt, wie Sie Daten des Programms inspizieren können. Nun geht's ans Ändern der Daten.

Gehen Sie mit den Pfeiltasten auf Zeile 114 in der Quelldatei. Plazieren Sie den Cursor auf die Variable `AnzahlBuchstaben` und drücken Sie [Ctrl][I], um den Wert dieser Variablen anzuschauen. Bei geöffnetem *Inspector*-Fenster drücken Sie nun [Alt][F10], um in das lokale Menü des *Inspector*-Fensters zu gelangen. Geben Sie C ein, um die Option *Change* anzuwählen (über [Ctrl][C] wäre dies auch möglich gewesen). Nun erscheint ein Dialogfenster, in dem Sie einen neuen Wert eingeben können.

An dieser Stelle können Sie jeden Pascal-Ausdruck verwenden, der eine Zahl ergibt. Schreiben Sie `AnzahlBuchstaben+4` und drücken Sie die Eingabetaste ↵. Im *Inspector*-Fenster erscheint der neue Wert 10.

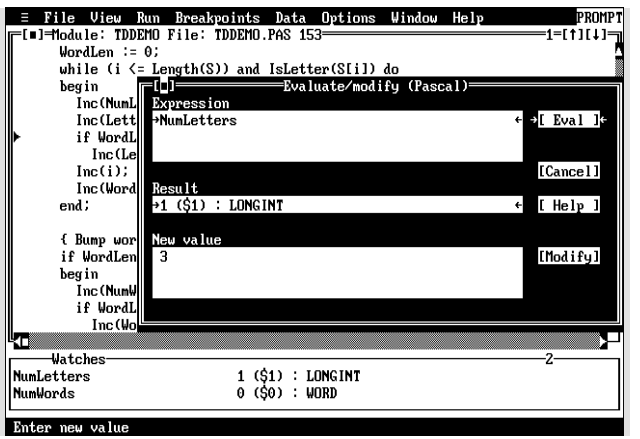


Abbildung 3.9 Das Dialogfenster von Evaluate/ Modify

Um Daten zu ändern, die nicht im *Module*-Fenster stehen, drücken Sie [Alt][D][E]. Es erscheint ein Dialogfenster, in dem der Name der zu ändernden Variablen einzugeben ist. Schreiben Sie `AnzahlZeilen` in das Eingabefeld und drücken Sie ↵. Das Ergebnis erscheint im mittleren Fensterausschnitt. Drücken Sie zweimal auf Tab, geben Sie 123 ein und drücken Sie die Eingabetaste ↵. Die Variable AnzahlZeilen enhält jetzt den Wert 123.

Damit sind wir am Ende unserer kurzen Einführung in die Untersuchung eines Turbo-Pascal-Programms mittels TDW. Kapitel 15, »Ein Standard-Pascal-Programm debuggen«, führt Sie durch ein ausführlicheres Beispiel.

4. Der Aufruf von Turbo Debugger

In diesem Kapitel erfahren Sie, wie Programme für das Debuggen vorbereitet werden. Zunächst beschreiben wir, wie Turbo Debugger unter Windows gestartet wird und wie man die zahlreichen Kommandozeilenoptionen des Debuggers entsprechend dem zu untersuchenden Programm einstellt. Weiterhin wird erklärt, wie man diese Optionen in einer Konfigurationsdatei speichern kann. Abschließend zeigen wir, Sie nach getaner Arbeit wieder zu Windows zurückkehren.

Die Vorbereitung des Programms für das Debuggen

Wenn Sie Ihre Programme mit Turbo Pascal für Windows compilieren, können Sie den Compiler anweisen, sämtliche Debug-Informationen zu generieren. Haben Sie die einzelnen Module Ihres Programm ohne diese Informationen compilieren lassen, wird Turbo Pascal jedes Modul, das Sie mit TDW untersuchen wollen, neu compilieren, wobei die notwendigen Informationen für den Debugger erzeugt werden.

Wenn Sie TDW direkt aus Windows aufrufen wollen und die Objektmodule Ihres Programms ohne Informationszusätze für den Debugger compiliert haben, dann müssen Sie alle Module zuerst neu compilieren, um für das ganze Programm die Möglichkeit des Debuggens auf Quelltextebene zu schaffen. Sie können die Erzeugung der Debug-Informationen auf spezielle Module beschränken, aber es kann später recht ärgerlich sein, wenn man beim Debuggen in ein Modul gerät, das keine Debug-Information enthält. Deshalb empfehlen wir Ihnen, alle Module erneut zu compilieren.

Arbeiten Sie mit der integrierten Entwicklungsumgebung von Turbo Pascal für Windows, müssen Sie folgende Einstellungen vornehmen, damit Debug-Informationen in der EXE-Datei Ihres Programms gespeichert werden:

- Wählen Sie die Menüoption *Options/Linker* und aktivieren in dem darauf geöffneten Dialogfenster *Linker Options* den Schalter *Debug Info in EXE*.
- Wählen Sie die Menüoption *Options/Compiler* und aktivieren in dem darauf geöffneten Dialogfenster *Compiler Options* den Schalter *Debug Information*. Sie

können aber auch den Compilerbefehl {$D+} an den Anfang des Quelltextes setzen.

- Wenn der Debugger auch auf lokale Symbole zugreifen soll (also Bezeichner, die innerhalb von Prozeduren oder Funktionen deklariert wurden), müssen Sie im Dialogfenster *Compiler Options* auch den Schalter *Local Symbols* aktivieren oder aber den Compilerbefehl {$L+} an den Anfang des Quelltextes setzen.

  Wenn Sie TPCW, die Kommandozeilen-Version des Compilers benutzen, müssen Sie die Option **/v** angeben. Genäß Voreinstellung werden dann Debug-Informationen zusammen mit den Informationen über lokale Symbole erzeugt. Deren Erzeugung können Sie mittels der entsprechenden **/$**-Option verhindern.

Den Debugger starten

Für den Aufruf von Turbo Debugger für Windows stehen Ihnen vier Wege offen:

- In der integrierten Entwicklungsumgebung von Turbo Pascal für Windows wählen Sie die Menüoption *Run/Debugger*. Dann können Sie das Programm im aktiven Editorfenster debuggen.
- Im Programm-Manager von Windows wählen Sie die Gruppe, zu der Turbo Debugger gehört, und klicken dessen Symbol zweimal an. Laden Sie dann über *File/Open* das Programm, das Sie untersuchen wollen.
- Wenn Sie zusätzlich Optionen in der Kommandozeile eingeben wollen, müssen Sie Programm-Manager die Menüoption *Datei/Ausführen...* aktivieren, worauf Windows das Dialogfenster *Ausführen* öffnet. Geben Sie in dem Eingabefeld TDW ein, gefolgt von den gewünschten Optionen und möglicherweise auch dem Namen des zu untersuchenden Programms.
- Von der DOS-Ebene aus können Sie TDW durch folgende Eingabe aufrufen:

```
WIN TDW [Optionen] [Programmname [Programmargumente]]
```

Der Kommandozeilenaufruf von Turbo Debugger

Wenn Sie TDW von der DOS-Ebene oder über *Datei/Ausführen...* im Programm-Manager aufrufen, können Sie Kommandozeilenoptionen direkt hinter dem Programmnamen TDW eingeben.

Diese Möglichkeit haben Sie nicht, wenn Sie den Debugger aus Turbo Pascal heraus starten. Sie können jedoch solche Optionen in der Datei TPW.INI angeben.

Kommandozeilenoptionen direkt eingeben

Die Syntax der Kommandozeile lautet:

`TDW [`*`Optionen`*`] [`*`Programmname`* `[`*`Programmargumente`*`]]`

Die Angaben in Klammern sind optional. Wenn Sie solche Angaben verwenden, schreiben Sie sie bitte ohne Klammern. *Programmname* ist der Name des Programms, das überprüft werden soll. Hinter dem Programmnamen können zusätzlich einige Argumente für das Programm folgen. Hier sehen Sie einige Beispiele für Kommandozeilenaufrufe:

Befehl	Aktion
`tdw -tc:\prog1 a b`	Startet den Debugger im Verzeichnis C:\PROG1 und lädt das Programm prog1 mit den zwei Argumenten a und b in der Kommandozeile.
`tdw prog2 -x`	startet den Debugger mit den Standardoptionen und lädt das Programm prog2 mit dem Argument -x in der Kommandozeile.

Wenn Sie nur TDW eingeben, wird nur der Turbo Debugger mit den Standard-Optionen geladen.

Kommandozeilenoptionen in TPW.INI angeben

Wenn Sie den Debugger von Turbo Pascal aus aufrufen und ihm einige Kommandozeilenoptionen mitgeben oder den Zugriffspfad zu TDW.EXE ändern wollen, müssen Sie dies in der Datei TPW.INI angeben. Sie finden diese Datei in dem Verzeichnis, in dem auch Windows selbst gespeichert ist.

Sie werden, wenn Sie diese Datei editieren, den Abschnitt *Startup* gefolgt von einer oder mehreren Angaben finden. Zum Beispiel:

```
[Startup]
CfgPath=C:\TPW\TPW.CFG
SizeOrg=2,-4,648,488
```

Nach dem *Startup*-Abschnitt können Sie weitere, den Debugger betreffende Informationen in die Datei einfügen. Dazu müssen Sie den neuen Abschnitt *Debugger* angeben, der zwei Einstellungen enthalten kann: *Exepath* und *Switches*. Dazu muß das folgende Format benutzt werden:

```
[Debugger]
Exepath=<Pfadname>
Switches=<Kommandozeilenoptionen>
```

Exepath
Sie müssen hier den Zugriffspfad nur angeben, wenn Sie TDW in einem anderen Verzeichnis gespeichert haben als dem, in das das Installationsprogramm TDW urprünglich kopiert hat.

Switches
Geben Sie hier, durch Leerzeichen getrennt, die Kommandozeilenoptionen an, nicht jedoch den Namen des zu untersuchenden Programms. Diesen Namen fügt Turbo Pascal automatisch hinzu.

Wenn Sie zum Beispiel TDW im Verzeichnis C:\TDEBUG gespeichert haben und TDW im Assembler-Modus starten wollen, wobei die Quelldateien im Verzeichnis C:\MYAPP\SOURCE zu suchen sind, muß der Abschnitt *Debugger* so aussehen:

```
[Debugger]
Exepath=C:\TDEBUG
Switches=-l -sd c:\myapp\source
```

Zur Wiederholung

Bitte beachten Sie, daß Sie bei der Arbeit mit Turbo Debugger sowohl die ».EXE«-Datei als auch alle verfügbaren originalen Quelldateien brauchen. Turbo Debugger sucht zuerst in dem Verzeichnis nach Quelldateien, in dem der Compiler sie beim Compilieren gefunden hat, zweitens in dem Verzeichnis, das unter *Options/Path for Source* angegeben wurde, drittens im aktuellen Verzeichnis und viertens in dem Verzeichnis, in dem sich die ».EXE«-Datei befindet.

- Der Quellcode muß bereits zusammen mit allen benötigten Debug-Informationen in eine Programmdatei (».EXE«) compiliert worden sein, bevor Sie mit dem Debuggen in TDW beginnen können.
- Turbo Debugger funktioniert nur mit Programmen, die mit Turbo Pascal für Windows compiliert wurden.
- Wenn Sie Ihr Programm unter Windows aufrufen und einen Fehler feststellen, müssen Sie Ihr Programm verlassen und in den Debugger laden, bevor Sie die Fehlersuche aufnehmen können.

Turbo Debugger und Windows

Wenn Sie TDW starten, werden Sie feststellen, daß der Dubugger nicht im Grafik- sondern nur im Textmodus läuft. Dabei nimmt der Debugger den gesamten Bildschirm ein, beschränkt sich also nicht wie andere Windows-Programme auf einen Ausschnitt des Bildschirms. Trotz dieser Erscheinungsform ist TDW ein Windows-Programm und nur unter Windows lauffähig.

Jedoch sind im Debugger die Windows-üblichen Tastenkürzel, wie [Alt][Esc] oder [Ctrl][Esc], nicht verfügbar. Wenn jedoch das Programm, das Sie untersuchen wollen, aktiv ist, stehen Ihnen auch diese Tastenkürzel zur Verfügung, über die Sie zu anderen Programmen wechseln können.

Hinweis: Wenn Sie mittels [Ctrl][Esc] ein unter Windows aktives Programm verlassen, erscheint dessen Name in der Task-Liste. In dieser Liste erscheint aber niemals der Name des Debuggers, denn TDW ist keine normale Windows-Task, in die Sie über die Task-Liste wechseln und wieder verlassen können.

Kommandozeilen-Optionen

Alle Kommandozeilenoptionen beginnen mit einem Minuszeichen (-) und sind vom Befehl TDW und untereinander durch mindestens ein Leerzeichen getrennt. Wenn Sie eine Kommandozeilenoption abschalten wollen, können Sie das durch die Eingabe eines weiteren Minuszeichens nach der Option. Zum Beispiel deaktiviert -p- die Maus. Eine solche Abschaltung ist dann sinnvoll, wenn eine Option in der Konfigurationsdatei aktiviert wurde. Die Konfigurationsdatei können Sie modifizieren, indem Sie das Konfigurationsprogramm TDWINST verwenden, das in Anhang C »Die Installation von TDW« beschrieben ist.

Im folgenden werden alle verfügbaren Kommandozeilenoptionen beschrieben. Anhang A »Kommandozeilenoptionen« enthält eine übersichtliche Aufstellung dieser Optionen.

Laden der Konfigurationsdatei (-c)

Durch diese Option wird die angegebene Konfigurationsdatei geladen. Zwischen -c und dem Dateinamen darf kein Leerzeichen stehen.

Wenn die Option -c nicht angegeben wird, lädt Turbo Debugger die Datei TDCONFIG.TD, falls diese vorhanden ist. Zum Beispiel:

```
TDW -cMEINCFG.TD TCDEMO
```

Damit wird die Datei MEINCFG.TDW zusammen mit dem Quellcode des Programms TPDEMO geladen.

Bildschirmumschaltung (-d)

Alle Optionen, die durch -d eingeleitet werden, beeinflussen die Art und Weise, wie die Umschaltung des Bildschirms durchgeführt wird.

-do Diese Option läßt den Debugger auf Ihrem zweiten Bildschirm laufen. Betrachten Sie den Bildschirm Ihres Programms auf dem ersten Monitor und lassen Sie den Debugger auf dem zweiten laufen.

-ds Die Kommandozeilenoption -ds ist die Standardoption (genannt *screen swapping*) für alle Bildschirme. Sie ist zwingend für Monochrombildschirme. Mit dieser Option wird eine separate Bildschirmoberfläche für den Debugger reserviert und eine andere für das Programm, das zu debuggen ist. Der ganze Bildschirm wird jedesmal aus dem Speicher geladen, wenn zwischen Programm und Debugger umgeschaltet wird. Dies ist zwar die zeitaufwendigste Methode für die Präsentation der beiden Bildschirmoberfächen, dafür ist sie sowohl auf allen Bildschirmen anwendbar als auch mit Programmen, die auf ungewöhnliche Art mit dem Bildschirm arbeiten.

Hilfestellungen (-h und -?)

Mit diesen Optionen erhalten Sie einen Bildschirm mit Hilfsinformationen, in denen die Kommandozeilenoptionen und die Kommandozeilensyntax von Turbo Debugger beschrieben werden.

Assembler-Modus (-l)

Mit dieser Option wird Turbo Debugger im Assembler-Modus gestartet. Der Startbildschirm enthält das *CPU*-Fenster. Turbo Debugger führt den Startup-Code Ihres Programms nicht aus, der normalerweise sofort beim Laden abgearbeitet wird. Das bedeutet, daß Sie auch den Initialisierungscode untersuchen können.

Wenn Sie eine dynamische Linkbibliothek debuggen, erlaubt Ihnen diese Option, den Initialisierungscode der DLL zu untersuchen. In Kapitel 11 »Debuggen unter Windows« gehen wir auf die Untersuchung von DLLs näher ein.

Maus-Unterstützung (-p)

Die Option `-p` schaltet die Mausunterstützung ein. Da Turbo Debugger standardmäßig auf Mausunterstützung eingestellt ist, brauchen Sie diese Option normalerweise nur dann anzugeben, wenn Sie die Maus abschalten wollen (-p-).

Hinweis: Wenn die Maus unter Windows nicht verfügbar ist, unterstützt sie auch TDW nicht. Die Kommandozeilenoption -p hat dann keine Wirkung.

Quellcode-Optionen (-s)

Alle Optionen, die mit -s beginnen, beeinflussen die Art und Weise, in der Turbo Debugger Quelldateien und Programmsymbole verarbeitet.

-sc Die Option -sc ignoriert Groß- und Kleinschreibung bei der Eingabe von Symbolnamen, auch wenn Ihr Programm unter Berücksichtigung von Groß- und Kleinschreibung gelinkt wurde.

Ohne die Option -sc ignoriert Turbo Debugger Groß- und Kleinschreibung nur dann, wenn Sie Ihr Programm mit der Option *Case ignore* gelinkt haben.

-sd Mit dieser Option können Verzeichnisse angegeben werden, in denen die Quelldateien liegen. Die Syntax ist:

```
-sdVerzeichnis
```

Um verschiedene Verzeichnisse zu wählen, verwenden Sie die Option -sd mehrmals. Mit jeder Option -sd kann nur ein Verzeichnisname angegeben werden. Die Verzeichnisse werden in der angegebenen Reihenfolge durchsucht. Der Suchpfad von *Verzeichnis* kann relativ oder absolut angegeben werden und auch eine Laufwerksangabe beinhalten. Wenn in der Konfigurationsdatei Verzeichnisse angegeben werden, so werden die Verzeichnisangaben, die mit Hilfe der Option -sd gewählt wurden, an diese Einträge angehängt.

Startverzeichnis (-t)

Über diese Option wird das Startverzeichnis angegeben, also jenes Verzeichnis, in dem TDW die Konfigurationsdatei sucht und auch jene EXE-Dateien, für die kein Zugriffspfad angegeben wurde. Zwischen Option und Zugriffspfad darf kein Leerzeichen eingegeben werden.

-t<Verz>*N* legt <Verz> als Startverzeichnis fest. Die Syntax ist:

```
-tVerzeichnis
```

Mittels dieser Option kann nur ein Verzeichnis angegeben werden. Ist sie in der Kommandozeile mehrere Male angegeben, wird die letzte Angabe verwendet. Zum Beispiel benutzt TDW bei folgender Angabe

```
tdw -tc:\utils\screensv -td:\working
```

das Verzeichnis D:\WORKING als Startverzeichnis.

Konfigurations-Dateien

Turbo Debugger verwendet eine Konfigurationsdatei, die die Voreinstellungen für die Kommandozeilenoptionen enthält. Sie können TDWINST zur Einstellung der Standardwerte für den Fall verwenden, daß keine Konfigurationsdatei vorhanden ist und eine Konfigurationsdatei erstellt werden soll.

Turbo Debugger sucht die Konfigurationsdatei TDCONFIG.TDW zuerst im aktuellen Verzeichnis, dann in dem Verzeichnis, das in TDWINST angegeben wurde, und schließlich in dem Verzeichnis, aus dem TDW.EXE aufgerufen wurde.

Wenn der Debugger eine Konfigurationsdatei findet, werden die dort angegebenen Einstellungen verwendet. Jede Kommandozeilenoption, die beim Aufruf von Turbo Debugger angegeben wird, hat jedoch den Vorrang gegenüber den Voreinstellungen. (In Anhang C, »Die Installation von TDW«, wird beschrieben, wie Sie mit dem Installationsprogramm eine Konfigurationsdatei erstellen können.)

Das Menü Options

Mit diesem Menü können Sie eine Anzahl von Parametern festlegen, die das allgemeine Erscheinungsbild und die Arbeitsweise von Turbo Debugger kontrollieren. Die folgenden Abschnitte beschreiben alle Menübefehle und verweisen Sie, wenn nötig, auf andere Abschnitte des Handbuchs mit weiteren Informationen.

Das Auswahlfeld Language

In Kapitel 9, »Ausdrücke«, wird beschrieben, wie man die Programmiersprache für die Auswertung der Ausdrücke einstellt und wie dieser Befehl die Art und Weise beeinflußt, in der Ausdrücke eingegeben werden müssen.

Das Menü Macros

Dieser Befehl aktiviert ein weiteres Menü, mit dessen Hilfe Sie neue Tastaturmakros definieren können, oder er löscht Makros, die Sie bereits einer Taste zugeordnet hatten. Das Menü enthält die folgenden Befehle: *Create*, *Stop recording*, *Remove* und *Delete All*.

Create

Mit dem Befehl *Options/Macros/Create* (engl. erzeugen) schalten Sie die Aufzeichnung von Tastatureingaben ein, die einer Taste zugeordnet werden sollen. Die Eingabeaufforderung fragt Sie zuerst nach der Taste, der das Makro zugeordnet werden soll, z.B. ([Alt][M]). Solange die Aufzeichnungsfunktion aktiv ist, erscheint in der oberen rechten Ecke des Bildschirms die Meldung Recording. Dann geben Sie die Tastenfolge ein, die Sie aufnehmen wollen. Diese Eingaben werden von Turbo Debugger genauso bearbeitet, als ob Sie kein Macro abspeichern würden.

Wenn Sie mit der Eingabe der Tastenfolge fertig sind, geben Sie den Befehl *Options/Macros/Stop recording* ein, oder das entsprechende Tastenkürzel [Alt][-]. Sie können aber auch die Tastenkombination eingeben, der das Makro zugeordnet wurde (in diesem Fall [Alt][M]).

Die Kombination [Alt][=] ist der Kurzcode, mit dem Sie die Aufnahme eines Makros starten können.

Stop recording

Durch den Befehl *Stop recording* (engl. Aufnahme beenden) wird die Aufzeichnung von Tastenkombinationen, die einer Taste zugeordnet werden, beendet. Verwenden Sie ihn, nachdem Sie über *Options/Macros/Create* die Aufnahme eines Makros gestartet haben und nun diese Aufnahme beenden wollen. Die Aufnahme des Makros wird auch durch den Kurzcode [Alt][-] beendet.

Remove

Mit dem Befehl *Remove* (engl. entfernen) können Sie ein Makro löschen, das einer Taste zugeordnet ist. Die Eingabeaufforderung fragt nach der Taste, deren zugeordnetes Makro Sie entfernen wollen.

Delete All

Der Befehl *Delete all* (engl. alle löschen) entfernt alle Makrodefinitionen für Tastenfolgen und stellt die urprünglichen Tastenbelegungen wieder her.

Das Dialogfenster Display options

Dieser Befehl öffnet ein Dialogfenster, in dem Sie verschiedene Optionen zur Bildschirmdarstellung von Turbo Debugger einstellen können.

Display swapping

Mit den Auswahlfeldern unter *Display Swapping* können Sie zwischen zwei Arten der Bildschirmauslagerung wählen:

Smart Es wird nur dann auf den Benutzerbildschirm umgeschaltet, wenn Ihr Programm eventuell etwas auf den Bildschirm ausgibt. Turbo Debugger wechselt die Bildschirme, sobald Sie eine Routine überspringen, oder wenn Sie einen Befehl oder eine Zeile ausführen, die unter Umständen aus dem Bildschirm-

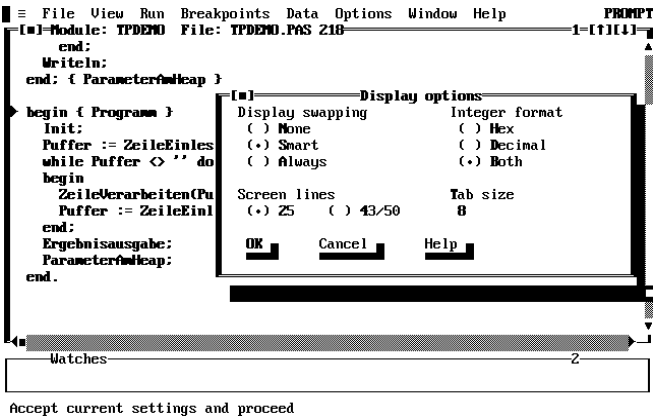


Abbildung 4.1 Das Dialogfenster Display Options

speicher liest oder dort hineinschreibt. Dies ist die Standardeinstellung.

Always Es wird immer auf den Benutzerbildschirm umgeschaltet, sobald das Benutzerprogramm läuft. Benützen Sie diese Option, wenn die Option *Smart* nicht bei allen Bildschirmausgaben Ihres Programms umschaltet. Unter dieser Option flackert Ihr Bildschirm jedesmal, wenn Sie eine Zeile in Ihrem Programm ausführen, da der Bildschirm von Turbo Debugger dabei kurz durch Ihren Programmbildschirm ersetzt wird.

Integer format

Die Option *Integer format* ermöglicht es Ihnen, zwischen drei Formaten der Integer-Darstellung zu wählen:

Decimal zeigt Integer-Zahlen als gewöhnliche Dezimalzahlen an.

Hex zeigt Integer-Zahlen als Hexadezimalzahlen an, die in einem Format dargestellt sind, das der eingestellten Programmiersprache entspricht.

Both zeigt Integer-Zahlen, sowohl als Dezimal- als auch als Hexadezimalzahlen an, die dann hinter dem dezimalen Wert in Klammern stehen.

Screen lines

Benützen Sie die Option *Screen lines* (engl. Bildschirmzeilen), um festzulegen, ob der Bildschirm von Turbo Debugger mit dem normalen 25-Zeilen-Modus oder mit den 43- oder 50-Zeilen-Modi arbeitet, die auf EGA- und VGA-Bildschirmadaptern zur Verfügung stehen.

Tab Size

In diesem Eingabefeld können Sie festlegen, wieviele Spalten ein Tabulator umfassen soll. Sie können die Breite der Tabulatorspalten verringern, um mehr

Text in Quelldateien darstellen zu können, die mit vielen Einrückungen versehen sind. Die mögliche Spaltenbreite liegt zwischen 1 und 32.

Der Befehl Path for source

Mit diesem Befehl werden die Verzeichnisse festgelegt, in denen Turbo Debugger nach Ihren Quelldateien sucht. In Kapitel 8, »Programmdaten untersuchen und ändern«, finden Sie im Rahmen der Beschreibung des *Module*-Fensters weitere Informationen zu dieser Option.

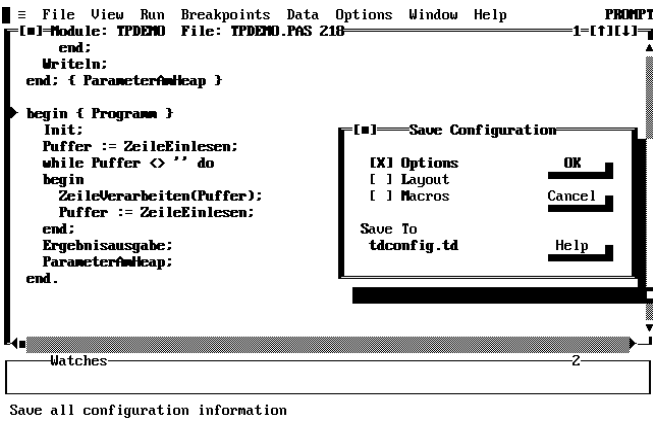


Abbildung 4.2 Das Dialogfenster Save Options

Der Befehl Save options

Dieser Befehl speichert die Optionen, die Sie eingegeben haben, in einer Konfigurationsdatei auf dem Laufwerk. Dazu zählen

- Ihre Makros,
- die aktuelle Fensteraufteilung,
- die Formate von Fensterausschnitten,
- alle Einstellungen, die im Menü *Options* vorgenommen worden sind.

Turbo Debugger läßt Ihnen die Wahl zwischen drei Möglichkeiten, Ihre Einstellungen zu sichern, je nachdem, welche Markierungsfelder unter *Save Configuration* angekreuzt wurden:

Options	speichert alle Einstellungen des Menüs *Options*.
Layout	speichert nur die Anordnung der Fenster.
Macros	speichert nur die aktuell definierten Makros.

Sie können auch das Eingabefeld *Save to* verwenden, um einen anderen Namen für die Konfigurationsdatei anzugeben, in der die Optionen gespeichert werden sollen.

Der Befehl Restore options

Damit können Sie die zuvor von Ihnen abgespeicherten Einstellungen aus einer Datei lesen. Das können verschiedenartige Konfigurationsdateien sein, die verschiedene Makros, Fensteraufteilungen usw. enthalten. Sie müssen dabei eine Datei angeben, die mit dem Befehl *Save Options* oder von TDWINST erzeugt wurde.

Rückkehr zu Windows

Sie können jederzeit Ihre Debugger-Sitzung beenden (außer, Sie befinden sich in einem Dialogfenster) und zu Windows zurückkehren, indem Sie [Alt][X] eingeben. Sie erreichen dasselbe, wenn Sie den Debugger mit *File/Quit* verlassen. (In einem Dialogfenster müssen Sie vorher Esc drücken.

5. Steuerung der Programmausführung

Will man ein Programm debuggen, geht man normalerweise so vor, das Programm stückweise auszuführen und dann zu überprüfen, ob es sich an einem bestimmten Punkt richtig verhält. Turbo Debugger bietet Ihnen viele Möglichkeiten, die Ausführung Ihres Programms zu kontrollieren. Sie können

- einzelne Maschinensprachebefehle oder einzelne Quellzeilen ausführen;
- Unterprogrammaufrufe überspringen;
- den Debugger »animieren« (kontinuierlich Trace-Befehle ausführen);
- eine aktuelle Prozedur oder Funktion so lange laufen lassen, bis sie zum aufrufenden Programm zurückkehrt;
- eine Funktion oder Prozedur bis zu einem bestimmten Punkt laufen lassen;
- weitermachen, bis ein Breakpoint erreicht wird;
- die Programmausführung rückgängig machen.

Eine Debuggersitzung besteht aus abwechselnden Phasen, in denen ent-weder Ihr Programm oder der Debugger läuft. Wenn der Debugger läuft, können Sie Ihr Programm starten, indem Sie eine Befehlsoption aus dem Menü *Run* wählen oder den entsprechenden Kurzcode eingeben. Läuft dagegen Ihr Programm, kann der Debugger die Kontrolle wieder übernehmen, wenn zum Beispiel ein spezieller Abschnitt Ihres Programms ausgeführt ist, oder wenn Sie die Ausführung mit einer speziellen Tastenfolge unterbrechen, oder schließlich, wenn Turbo Debugger auf einen Breakpoint trifft.

Dieses Kapitel zeigt Ihnen, wie Sie den Zustand Ihres Programms untersuchen können, wenn der Debugger arbeitet. Sie lernen verschiedene Methoden kennen, um Teile Ihres Programms auszuführen bzw. während der Ausführung zu unterbrechen. Schließlich erfahren Sie, wie Sie eine neue Debugger-Sitzung starten, sowohl mit demselben als auch mit einem anderen Programm.

Den aktuellen Zustand des Programms untersuchen

Der »Zustand« Ihres Programms besteht aus folgenden Elementen:

- seinen Kommandozeilenargumenten;
- dem Stack mit den aktiven Funktionen und Prozeduren;
- der aktuellen Position im Quell- oder Maschinencode;
- den Registerwerten;
- dem Speicherinhalt;
- dem Grund, aus dem Debugger Ihr Programm gestoppt hat;
- den Werten der Variablen Ihres Programms.

Die folgenden Abschnitte erklären das *Variables*-Fenster, das *Stack*-Fenster, die lokalen Menüs der globalen und statischen Fensterausschnitte, den Befehl *Origin* und den Befehl *Get Info*. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen darüber, wie Sie Werte von Variablen in Ihrem Programm untersuchen und ändern können.

Das Variables-Fenster

Dieses Fenster zeigt alle Variablen (Namen und Werte), die Sie von der aktuellen Position in Ihrem Programmm aus erreichen können. Verwenden Sie es, um Variablen zu finden, an deren Schreibweise Sie sich nicht erinnern können. Dann können Sie ihre Werte mit Hilfe der Befehle aus dem lokalen Menü näher untersuchen oder ändern. Sie können dieses Fenster auch dazu verwenden, die lokalen Variablen zu allen Funktionen, die aufgerufen werden, zu untersuchen.

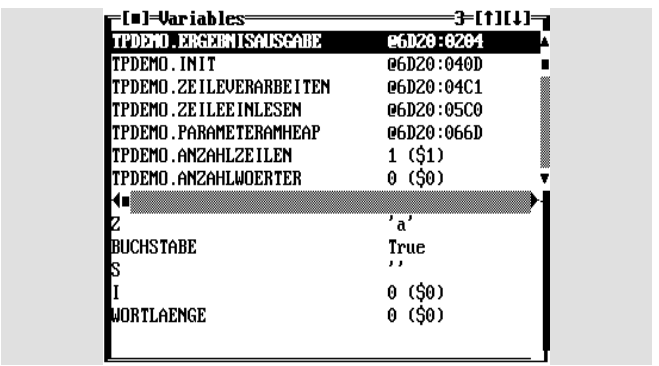


Abbildung 5.1 Das Variables-Fenster

Achtung: Wenn Sie ein Turbo-Pascal-für-Windows-Programm debuggen, sind die Variablen nicht alphabetisch geordnet.

Sie öffnen das Fenster *Variables*, indem Sie über die Menüleiste den Befehl *View / Variables* auswählen. Ein *Variables*-Fenster hat zwei Ausschnitte:

- einen globalen Ausschnitt (oben) mit allen globalen Symbolen Ihres Programms;
- einen statischen Ausschnitt (unten) mit den statischen Symbolen des aktuellen Moduls (das Modul mit der aktuellen Programmposition, CS:IP) und den lokalen Symbolen der aktuellen Funktion.

In beiden Ausschnitten steht der Name der Variablen am linken Rand und ihr Wert am rechten Rand. Wenn Turbo Debugger keine Informationen für das Symbol findet, erscheinen vier Fragezeichen (????).

Wie alle lokalen Menüs, können Sie auch das lokale Menü des globalen Ausschnitts durch die Eingabe von [Alt][F10] aufrufen. Falls die Kurzcodes über die [Ctrl]-Taste aktiviert sind, können Sie [Ctrl] in Kombination mit dem ersten Buchstaben des gewünschten Befehls drücken, um diesen auszuwählen.

Wenn Ihr Programm Funktionen mit rekursiven Aufrufen enthält oder wenn Sie die lokalen Variablen zu einer aufgerufenen Funktion anschauen möchten, können Sie die lokalen Daten einer Funktion untersuchen. Rufen Sie zunächst über *View / Stack* das *Stack*-Fenster auf, bewegen Sie dann den Cursor auf den gewünschten Punkt im Funktionsaufruf. Danach drücken Sie [Alt][F10] und wählen *Locals*. Der statische Ausschnitt aus dem *Variables*-Fenster zeigt nun die Werte für diesen speziellen Punkt der Funktion.

Das lokale Menü des globalen Fensterausschnitts

Dieses lokale Menü umfaßt drei Befehle: *Inspect*, *Change* und *Watch*.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein *Inspector*-Fenster, das den Inhalt des momentan unterlegten globalen Symbols zeigt. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen über die Arbeit mit *Inspector*-Fenstern.

Wenn die Variable, die Sie untersuchen wollen, der Name einer Routine ist, wird der Quellcode für diese Funktion gezeigt. Falls keine Quelldatei existiert, zeigt ein *CPU*-Fenster den disassemblierten Code.

Wenn sich der Name der untersuchten Variablen mit einer lokalen Variablen des gleichen Namens überschneidet, so sehen Sie nicht den Wert der lokalen Variablen, sondern den der globalen. Üblicherweise verhalten sich *Inspector*-Fenster etwas anders: Sie zeigen gewöhnlich den Wert einer Variablen innerhalb des aktuellen Bereichs im Programm (CS:IP). Durch das veränderte Verhalten haben Sie eine bequeme Möglichkeit, sich die Werte von globalen Variablen anzuschauen, deren Namen auch für lokale Variablen verwendet werden.

Change

Der Befehl *Change* (engl. verändern) verändert den Wert des momentan ausgewählten Symbols auf den Wert, der in einem Dialogfenster neu eingeben

wird. Turbo Debugger führt alle notwendigen Umwandlungen von Datentypen genauso aus, als würden Sie durch den jeweiligen Zuweisungsoperator der Sprache vollzogen, mit der Sie gerade arbeiten. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über Zuweisungen und Veränderungen von Datentypen.

Sie können den Wert der gerade ausgewählten Variablen auch einfach dadurch ändern, daß Sie anfangen, einen neuen Wert einzugeben. Dadurch erscheint das gleiche Dialogfenster, das auch aktiviert wird, wenn Sie vorher den Befehl *Change* eingeben.

Watch

Der Befehl *Watch* öffnet ein *Watches*-Fenster und stellt darin das momentan selektierte (unterlegte) globale Symbol dar. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie mehr Informationen zum Fenster *Watches*.

Das *Watches*-Fenster verfolgt nicht, ob eine Variable global oder lokal ist. Wenn Sie eine globale Variable mit dem *Watch*-Befehl einfügen und später auf eine lokale Variable mit demselben Namen treffen, überwiegt die lokale Variable solange Sie sich in ihrem Block befinden. Mit anderen Worten zeigt Ihnen das *Watches*-Fenster immer den Wert einer Variablen aus der Sicht der aktuellen Position in Ihrem Programm (CS:IP).

Das lokale Menü im statischen Fensterausschnitt

Geben Sie die Tastenfolge [Alt][F10] ein, um das lokale Menü des statischen Fensterausschnitts aufzurufen. Falls die Tastenkürzel im Zusammenhang mit der Ctrl-Taste aktiviert sind, gelangen Sie zu dem gewünschten Befehl, indem Sie die Ctrl-Taste zusammen mit dem ersten Buchstaben dieses Befehls drücken.

Der statische Fensterausschnitt verfügt über vier Menübefehle: *Inspect*, *Change, Watch* and *Show*.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein *Inspector*-Fenster, das den Inhalt des gerade ausgewählten lokalen Symbols zeigt. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen über die Arbeit mit *Inspector*-Fenstern.

Change

Der Befehl *Change* (engl. verändern) verändert den Wert des momentan ausgewählten lokalen Symbols auf den Wert, der in einem Dialogfenster neu eingegeben wird. Turbo Debugger führt alle notwendigen Unwandlungen von Datentypen genauso aus, als würden Sie durch den entsprechenden Zuweisungsoperator der Sprache vollzogen, mit der Sie gerade arbeiten. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über Zuweisungen und Veränderungen von Datentypen.

Abbildung 5.2 Das Dialog-Fenster Local Display

Sie können den Wert einer Variablen, die gerade ausgewählt ist, auch einfach dadurch ändern, daß Sie anfangen, einen neuen Wert einzugeben. Dadurch erscheint das gleiche Dialogfenster wie beim Befehl *Change*.

Watch

Der Befehl *Watch* öffnet eine *Watches*-Fenster und stellt das momentan selektierte (unterlegte) statische oder lokale Symbol hinein. In Kapitel 6 »Programmdaten untersuchen und ändern« den Sie mehr Informationen über das Arbeiten mit *Watches*-Fenstern.

Show

Der Befehl *Show* öffnet das Dialogfenster *Local Display* in dem Sie einstellen können, welche Variablen angezeigt werden (statische, auto oder beides) und aus welchem Modul diese Variablen kommen sollen.

Folgende Auswahlmöglichkeiten haben Sie in diesem Dialogfenster:

Static	Zeige nur statische Variablen
Auto	Zeige nur lokale Variablen des aktuellen Blocks
Both	Zeige Variablen beider Typen (Standardeinstellung)
Module	Wechsle das aktuelle Modul. Diese Auswahl öffnet ein Dialogfenster mit der Modulliste des Programms, aus der Sie ein neues Modul auswählen können.

Das Fenster Stack

Ein *Stack*-Fenster kann mit dem Befehl *View / Stack* angelegt werden. Das *Stack*-Fenster enthält eine Liste mit allen aufgerufenen und aktiven Funktionen oder Prozeduren. Die zuletzt aufgerufene Routine wird in der Liste als erste dargestellt. Dahinter folgen die aufrufende Routine und dann deren aufrufende Prozedur, bis man zur der ersten Funktion oder Prozedur des Programms kommt (in Pascal das Hauptprogramm). Zu jeder Funktion oder Prozedur sehen Sie die Werte der einzelnen Parameter, mit denen die Routine aufgerufen wurde.

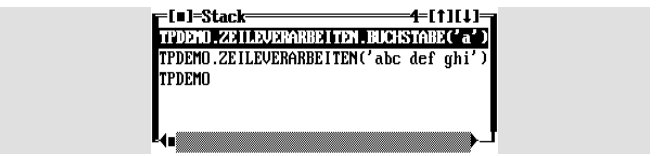


Abbildung 5.3 Das Stack-Fenster

Im *Stack*-Fenster erscheinen die Namen der Methoden oder Elementfunktionen hinter der Angabe der entsprechenden Objekte oder Klassen, in denen die Methode oder Elementfunktion definiert ist:

```
GRAFIKEN.KREIS(174, 360, 75,0)   { Turbo Pascal }
```

Wenn Sie das lokale Menü des *Stack*-Fensters aktivieren wollen, drücken Sie die Tastenkombination [Alt][F10]. Mit Ctrl und dem Anfangsbuchstaben des jeweiligen Befehls können die Befehle dieses Menüs auch direkt aufgerufen werden.

Das lokale Menü des Stack-Fensters
Dieses Menü stellt die beiden Befehle *Inspect* und *Locals* bereit.

Inspect
Mit dem Befehl *Inspect* (engl. untersuchen) wird ein *ModuleFenster ab der aktivierten Zeile in der gerade ausgewählten Funktion geöffnet. Wenn die hervorgehoben dargestellte Funktion ganz oben steht (also zuletzt aufgerufen wurde), zeigt das Module*-Fenster die aktuelle Programmposition (CS:IP). Gehört die ausgewählte Funktion zu einer der Funktionen, durch die die letzte Funktion aufgerufen haben, so wird der Cursor auf die Zeile im Fenster gesetzt, in der der nächste Befehl hinter dem Funktionsaufruf steht.

Dieser Befehl kann auch mit der Eingabetaste ↵ aufgerufen werden, wenn die gewünschte Funktion hervorgehoben dargestellt ist.

Locals
Der Befehl *Locals* (engl. lokale [Variablen]) öffnet ein *Variables*-Fenster, in dem sowohl die lokalen Symbole des aktuellen Moduls als auch die Symbole der gerade ausgewählten Funktion dargestellt werden. Wenn sich eine Funktion rekursiv selbst aufruft, erscheint sie auch mehrmals im *Stack*-Fenster. Falls Sie den Auswahlbalken auf einen bestimmten Funktionsaufruf setzen, können Sie die lokalen Variablen dieses Funktionsaufrufs untersuchen.

Der Befehl Origin im lokalen Menü

Das *Module*-Fenster und der Code-Ausschnitt eines *CPU*-Fensters stellen beide den lokalen Befehl *Origin* (engl. Ursprung) in ihren lokalen Menüs bereit. Der Befehl *Origin* setzt den Cursor an die aktuelle Position des Programms (CS:IP). Das ist nützlich, wenn Sie im Code umhergewandert sind und wieder zu dem Punkt zurückwollen, an dem das Programm angehalten wurde.

Der Befehl Get Info

Mit dem Befehl *File / Get info* (engl. Informationen abrufen) erhalten Sie einen Überblick über die Speicherplatzbelegung und die Steuerung des Programms durch den Debugger. Diese und andere Informationen erscheinen in einem Informationsblock, der wieder entfernt wird, wenn Sie die Eingabetaste ↵, die [Leertaste] oder [Esc] drücken. Die folgenden Informationen erscheinen im Fenster Systeminformation:

der Name des Programms, das Sie debuggen;

- eine Beschreibung, wo und warum das Programm angehalten wurde;
- Informationen über die Speicherbereiche Ihres Systems.
- Datum und Uhrzeit.

Status line messages

Folgende Meldungen in der zweiten Zeile (Statuszeile) geben den Grund an, warum das Programm angehalten wurde:

Stopped at__
[Angehalten in__]
Ihr Programm wurde nach Ausführung der folgenden Befehle angehalten: *Run/Execute to*, *Run/Go to cursor* oder *Run/Until return*. In der Statuszeile wird außerdem dargestellt, wann das Programm zum ersten Mal geladen wurde und daß Ihr Programm den Startcode des Compilers bis zum Anfang Ihres Quellcode-Moduls ausgeführt hat.

No program loaded
[Kein Programm geladen]
Sie haben Turbo Debugger ohne ein Programm gestartet. Sie können keinen Code ausführen, bis Sie entweder ein Programm laden oder einige Befehle mit dem lokalen Menübefehl *Assemble* im Code-Ausschnitt des CPU-Fensters assemblieren.

Informationen über den globalen Speicher
TDW gibt Ihnen folgende Informationen über den globalen Speicher:

Mode	Folgende Speicher-Modi sind möglich: Large-frame EMS, Small frame EMS und non-EMS (Extended Memory).
Banked	Speicher in KByte oberhalb der EMS *Bank Line* (der zum Expanded Memory werden kann, wenn das System dieses verwendet).

Not banked	Speicher in KByte unterhalb der EMS *Bank Line* (der nicht zum Expanded Memory werden kann).
Largest	Der größte, zusammenhängende Speicherbereich in KByte.

Control Break
[Unterbrechung durch [Ctrl][Break]]
Sie haben die Ausführung Ihres Programms mit [Ctrl][Break] unterbrochen.

Trace
[Einzelschritt ausgeführt]
Sie haben eine einzelne Quellzeile oder einen Maschinenbefehl mit F7 oder *Run/Trace* ausgeführt.

Breakpoint at __
[Breakpoint in __]
Ihr Programm ist an einem Breakpoint angehalten worden, der von Ihnen vorher gesetzt wurde. Hinter »at« steht die Programmadresse des Breakpoints.

Loaded
Geladen
Sie haben Turbo Debugger mit einem Programm aufgerufen. *Außerdem* haben Sie eine Kommandozeilenoption angegeben, mit der verhindert wird, daß der Startcode des Compilers ausgeführt wird. Zu diesem Zeitpunkt wurden noch keine Befehle, auch nicht zur Initialisierung des Stacks oder des Segmentregisters, ausgeführt. Das bedeutet, daß Sie bei der Untersuchung von Programmdaten falsche Werte vorfinden werden.

Step
[Einzelschritt ausgeführt]
Sie haben einen einzelnen Befehl oder einen Maschinenbefehl mit F8 oder dem Befehl *Run/Step* ausgeführt, bei dem Unterprogrammaufrufe übersprungen werden.

Interrupt
Interrupt
Sie haben die Interrupt-Taste (normalerweise ist das [Ctrl][Break]) gedrückt, um wieder die Kontrolle des Programms zu übernehmen. Ihr Programm wird sofort unterbrochen, und der Debugger wird aktiviert.

Exception __
[Ausnahme __]
Während der Arbeit mit TD386 trat eine Ausnahmebedingung des Prozessors auf. Das passiert meistens, wenn in Ihrem Programm ein unzulässiger Befehl ausgeführt werden soll. In den Intel-Handbüchern sind die entsprechenden Ausnahmebedingungen ausführlich dokumentiert.

Divide by zero
[Division durch Null]
In Ihrem Programm wurde versucht, eine Division mit einem Divisor von Null durchzuführen.

Global breakpoint __ at __
[Globaler Breakpoint __ an Position __]
Es wurde ein globaler Breakpoint ausgelöst. Sie erhalten die Nummer des Breakpoints und die Stelle im Programm, an der der Breakpoint auftrat.

Window message breakpoint at __
[Fenstermeldungs-Breakpoint in __]
Ihr Programm ist an einem Fenstermeldungs-Breakpoint angehalten worden, der von Ihnen vorher gesetzt wurde. Hinter »at« steht die Fenster-Prozedur, für die die Meldung bestimmt war.

Terminated, exit code __
[Beendet, Abschlußscode __]
Das Programm ist bis zum Ende ausgeführt worden. Hinter »exit code« steht der Abschlußcode, der von Ihrem Programm an Windows zurückgegeben wird. Wenn Ihr Programm keinen bestimmten Wert zurückgibt, erhalten Sie ein zufälliges Ergebnis. Sie können Ihr Programm nicht mehr laufen lassen, bis Sie das Programm mit *Run/Program reset* erneut laden.

Das Menü Run

Das Menü *Run* enthält eine Reihe von Optionen, mit denen Sie verschiedene Abschnitte Ihres Programms ausführen können. Da diese Optionen sehr häufig gebraucht werden, sind sie alle über Funktionstasten abrufbar.

Run F9

Durch den Befehl *Run* (engl. starten) wird das Programm mit der vollen Ablaufgeschwindigkeit ausgeführt. Der Debugger erhält erst dann wieder die Kontrolle, wenn eines der folgenden Ereignisse eintritt:

- das Programm wird beendet;
- ein Breakpoint wird erreicht;
- Sie unterbrechen das Programm mit [Ctrl][Break].

Go to cursor F4

Mit dem Befehl *Go to cursor* (engl. Gehe bis zum Cursor) wird Ihr Programm bis zu der Zeile ausgeführt, in der der Cursor im aktuellen *Module*- oder *CPU*-Fenster steht. Wenn Sie sich gerade in einem *Module*-Fenster befinden, muß der Cursor auf einer Quellzeile in einer Funktion stehen.

Trace into F7

Eine einzelne Quellzeile oder einen Maschinensprachebefehl kann man mit dem Befehl *Trace into* (engl. nachspüren) ausführen. Normalerweise wird eine einzelne Quellzeile ausgeführt. Wenn die aktuelle Zeile einen Prozedur- oder Funktionsaufruf enthält, geht Turbo Debugger in diese Routine. Falls Sie jedoch in einem *CPU*-Fenster sind, wird nur ein einziger Assemblerbefehl ausgeführt.

Turbo Debugger behandelt Objektmethoden und Elementfunktionen einer Klasse wie jede andere Prozedur oder Funktion. Mit F7 gelangen Sie in den Quellcode, falls dieser vorhanden ist.

Step over F8

Der Befehl *Step over* (engl. schrittweise überspringen) führt eine einzelne Quellzeile oder einen Maschinensprachebefehl aus, übergeht jedoch alle Prozedur- und Funktionsaufrufe. Im Regelfall wird eine Quellzeile ausgeführt. Wenn Sie jedoch in einem *CPU*-Fenster sind, wird nur ein einzelner Maschinensprachebefehl ausgeführt.

Wenn Sie über eine einzelne Quellzeile springen, behandelt Turbo Debugger alle Funktions- oder Prozeduraufrufe in dieser Zeile als Bestandteil der Zeile, und Sie landen nicht am Anfang einer dieser Funktionen. Stattdessen kommen Sie direkt zur nächsten Zeile in der aktuellen Routine, oder in der Routine, die diese Routine aufgerufen hat.

Wenn Sie einen einzelnen Maschinenbefehl überspringen, interpretiert Turbo Debugger bestimmte Befehle als Einzelbefehle, auch wenn damit mehrere Befehle auf einmal ausgeführt werden. Die folgende Aufstellung enthält alle Befehle, die Turbo Debugger mit *Step over* als einen einzelnen Befehl verarbeitet:

- `CALL` Unterprogrammaufruf, `NEAR` oder `FAR`
- `INT` Interrupt-Aufruf
- `LOOP` Schleife mit CX als Zähler
- `LOOPZ` Schleife mit CX als Zähler
- `LOOPNZ` Schleife mit CX als Zähler

Es werden auch alle Kombinationen von REP, REPNZ oder REPZ mit CMPS, CMPSB, CMPSW, LODS, LODSB, LODSW, MOVS, MOVSW, SCAS, SCASB, SCASW, STOS, STOSB, STOSW als ein Einzelbefehl behandelt.

Der Befehl *Run/Step over* behandelt alle Objektmethoden oder Elementfunktionen einer Klasse als Einzelbefehl und übergeht sie wie alle anderen Prozedur- und Funktionsaufrufe.

Execute to... [Alt][F9]

Mit dem Befehl *Execute to* (engl. ausführen bis) wird das Programm bis zur angegebenen Adresse ausgeführt. Die Adresse wird unter Umständen gar nicht erreicht, wenn vorher ein Breakpoint ausgelöst oder das Programm von Ihnen unterbrochen wurde.

Until return [Alt][F8]

Nach der Eingabe des Befehls *Until return* (engl. bis zum Rücksprung) wird das Programm solange ausgeführt, bis die aktuelle Funktion zur aufrufenden Funktion zurückkehrt. Das ist vor allem in folgenden Situationen sehr praktisch: wenn Sie mit *Run/Trace* aus Versehen in eine Prozedur oder Funktion hineingegangen sind, die Sie eigentlich gar nicht interessiert, oder wenn Sie festgestellt haben, daß die aktuelle Funktion zu Ihrer Zufriedenheit funktioniert und Sie den Rest gar nicht mehr überprüfen wollen.

Animate...

Mit dem Befehl *Animate* (engl. animieren) wird das Programm langsam Schritt für Schritt ausgeführt, wobei alle Veränderungen nach jedem einzelnen Programmschritt dargestellt werden. Damit kann man die aktuelle Position im Quellcode und die Variablenwerte verfolgen. Dieser Befehl kann mit einem beliebigen Tastendruck unterbrochen werden.

Nachdem Sie *Run/Animate* ausgewählt haben, werden Sie nach der gewünschten Verzögerungsdauer zwischen zwei Programmschritten gefragt. Die Zeitverzögerung wird in Zehntelsekunden gemessen; der Standardwert beträgt 3.

Back trace

Wenn Sie ein Programm schrittweise ausführen (F7 oder [Alt][F7]) können Sie mit diesem Befehl die letzten Befehle zurückgehen. Dies ist sehr praktisch, wenn

Sie über den Punkt hinausgegangen sind, an dem Sie einen Fehler vermuten. Damit können Sie die Ausführung des Programms ungeschehen machen. Dies kann ein einzelner Programmschritt oder eine bestimmte Stelle sein, die im Befehlsausschnitt des Fensters *Execution History* ausgewählt ist.

Achtung: Es sind jedoch einige Einschränkungen zu beachten (siehe »Der Ausschnitt mit den ausgeführten Programmschritten« weiter hinten in diesem Kapitel).

Instruction trace [Alt][F7]

Der Befehl *Instruction trace* (engl. Befehlsschritt) führt einen einzelnen Maschinensprachebefehl aus. Verwenden Sie diese Option, wenn Sie einen Interrupt nachvollziehen wollen oder wenn Sie in einem *Module*-Fenster sind und eine Prozedur oder Funktion debuggen wollen, die keine Debug-Informationen enthält (zum Beispiel eine Bibliotheksroutine).

Da Sie sich nicht mehr am Anfang einer Quellzeile befinden, kommen Sie automatisch in ein *CPU*-Fenster.

Arguments...

Mit diesem Befehl können Sie andere Kommandozeilenargumente für Ihr Programm angeben. (Siehe »Die Argumente eines Programms ändern« am Ende dieses Kapitels)

Program reset [Ctrl][F2]

Der Befehl *Program reset* (engl. Programm zurücksetzen) lädt das Programm, das Sie debuggen, erneut vom Laufwerk. Verwenden Sie diesen Befehl,

- wenn Sie mit der Ausführung des Programms »zu weit« gegangen sind (d.h., wenn Sie die Fehlerstelle übersprungen haben).
- Wenn Ihr Programm beendet ist und Sie es noch einmal starten möchten.
- bei einer Windows-Anwendung, wenn Sie sich in einem *Module*- oder *CPU*-Fenster befinden, die Anwendung mit *Ctrl-Alt-Sysrq* angehalten haben und sie jetzt beenden und neu starten möchten.
- wenn Sie Ihre Windows-Anwendung schon geladen und Startup-Debuggen für eine oder mehrere dynamische Link-Bibliotheken (DLLs) eingestellt haben und Sie jetzt diese DLLs debuggen möchten.

Wenn Sie sich in einem *Module*- oder *CPU*-Fenster befinden, wird die Zeilenmarkierung auf den Anfang des Programms gesetzt, der Programmstart wird jedoch nicht dargestellt. Stattdessen bleiben Sie genau an der Stelle, an der Sie vor dem Befehl *Program reset* waren. Dadurch wird es für Sie leichter, den Cursor

dorthin zu positionieren, wo Sie vorher waren, und das Programm bis zu dieser Zeile laufen zu lassen.

Wenn Sie den Befehl *Program reset* ausgewählt haben, weil Sie gerade eine Anweisung in Ihrem Quelltext zuviel ausgeführt haben, können Sie den Cursor ein paar Zeilen im Quelltext zurücksetzen und F4 drücken, um das Programm bis zu dieser Position laufen zu lassen. Alternativ könnten Sie, anstatt *Program Reset* zu wählen, mit *Run/Backtrace* schrittweise durch den vorher ausgeführten Code zurückgehen, wenn Sie *Full History* (das lokale Fenster des Fensters *View/Execution History*) einstellen.

Das Fenster Execution History

Turbo Debugger bietet eine ganz spezielle Möglichkeit der Aufzeichnung aller ausgeführten Programmschritte in einem Fenster: die *Execution history*. In diesem Fenster können Sie außerdem alle Tastatureingaben erfassen, die Sie bis zu einer bestimmten Stelle im Programm eingegeben haben. Diese Anweisungen können zum einen untersucht werden und zum anderen dazu verwendet werden, an eine bereits übersprungene Programmstelle zurückzugehen, an der Sie einen Fehler vermuten. Turbo Debugger kann bis zu 400 Anweisungen aufzeichnen.

Sie können die Programmschritte im Fenster *Execution history* untersuchen, das sich mit dem Befehl *View/Execution history* öffnen läßt.

Dieses Fenster hat zwei Ausschnitte: den Befehlsausschnitt oben und einen Ausschnitt zum Aufzeichnen der Tastatureingaben.

Der Ausschnitt mit den ausgeführten Programmschritten

Der Befehlsausschnitt zeigt die Befehle, die bereits ausgeführt wurden und untersucht oder ungeschehen gemacht werden können. Benutzen Sie den Auswahlbalken zur Selektion.

Hinweis: In der *Execution history* werden nur die Programmschritte aufgezeichnet, die mit dem Befehl *Trace into* (F7) oder *Instruction Trace* (Alt F7) ausgeführt wurden. Außerdem werden die *Step over*-Befehle festgehalten, solange keiner der Befehle ausgeführt wird, die im Rahmen dieses Befehls angegeben sind. Sobald

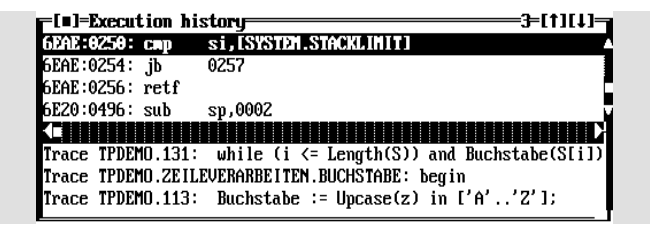


Abbildung 5.4
Das Fenster Execution History

Sie einen *Run*-Befehl ausführen oder einen Interrupt aufrufen, wird die *Execution history* zerstört. (Die Aufzeichnung beginnt wieder ab dem nächsten *Trace*-Befehl).

Achtung: Interrupt-Aufrufe lassen sich nicht zurückverfolgen. Wenn Sie über einen Prozedur- oder Funktionsaufruf hinweggehen, können Sie nicht mehr über den Befehl hinter dem Rücksprung zurückspringen. Sie können auch nicht über Befehle zurückgehen, die I/O-Ports benutzen, da Schreib- und Lesebefehle dort nicht mehr rückgängig gemacht werden können.

Das lokale Menü des Befehlsausschnitts

Im lokalen Menü des Befehlsausschnitts stehen drei Befehle: *Inspect*, *Reverse Execute*, *Full History*.

Inspect

Dieser Befehl führt Sie zu einem Befehl, der im Befehlsausschnitt hervorgehoben dargestellt ist. Wenn es sich um eine Quellzeile handelt, wird die entsprechende Zeile im *Module*-Fenster angezeigt. Falls kein Quellcode verfügbar ist, wird das *CPU*-Fenster geöffnet, in dem der Befehl im Code-Ausschnitt angezeigt wird.

Reverse Execute

Mit diesem Befehl können Sie das Programm rückwärts bis zu der Programmstelle ausführen, die im Befehlsausschnitt hervorgehoben dargestellt ist. Wenn Sie eine Quellcodezeile ausgewählt haben, kommen Sie ins *Module*-Fenster, sonst erscheint das *CPU*-Fenster, in dem der Programmschritt im Code-Ausschnitt hervorgehoben wird. (Das Tastenkürzel für diesen Befehl lautet [Alt][F4].)

Achtung: Sie können nicht in einem Programmabschnitt zurückgehen, der nicht per Einzelschritt abgearbeitet wurde. Wenn Sie zum Beispiel einen Breakpoint gesetzt haben und das Programm mit [F9] zu dieser Programmstelle laufen ließen, wird Ihre gesamte *Execution history* verworfen. In diesem Fall bleibt Ihnen zum Wiederherstellen der vorherigen Situation nur noch das Abspielen der Tastatureingaben, indem Sie das Programm erneut laden und bis zur gewünschten Stelle laufen lassen.

Durch die Ausführung eines INT-Befehls werden alle Aufzeichnungen in der *Execution history* verworfen. Über diesen Befehl kommen Sie nicht zurück, falls Sie nicht mit [Alt][F7] in den Interrupt eingestiegen sind. Durch die folgenden Befehle werden die Aufzeichnungen nicht gelöscht, aber die Wirkung kann nicht ungeschehen gemacht werden. Sie sollten auf Überraschungen gefaßt sein, wenn Sie über diese Befehle hinweg zurückgehen wollen.

```
IN        INSW
OUT       OUTSB
INSB      OUTSW
```

Full history

Mit diesem Befehl können Sie die Rückwärts-Ausführung des Programms ein- oder ausschalten.

Programmausführung unterbrechen

Bei interaktiven Programmen kommt man am schnellsten an eine bestimmte Stelle, indem man das Programm einfach laufen läßt, und so lange damit arbeitet, bis die gewünschte Stelle erreicht ist, an der man es dann abbricht. Das trifft vor allem dann zu, wenn der Programmabschnitt, der untersucht werden soll, mehrmals aufgerufen wird, bevor Sie sich speziell dafür interessieren.

Sie werden die Programmausführung auch dann unterbrechen müssen, wenn das Programm aus irgendeinem Grund nicht mehr zum Debugger zurückkehrt. Das passiert dann, wenn eine Endlosschleife programmiert wurde: Sie erwarten, daß ein bestimmter Programmteil ausgeführt wird, und legen einen Breakpoint fest, aber dieser Codebereich wird nie erreicht.

Ctrl-Break

Mit der Tastenkombination [Ctrl][Break] werden Sie Ihr Programm fast immer unterbrechen und zum Debugger zurückkehren können. Diese Tastenkombination wirkt sich sofort nach dem Drücken der Tasten aus, so daß Sie manchmal in einem nicht erwarteten Codeabschnitt landen. Dieser Code kann im ROM des Tastatur-BIOS liegen, wenn Ihr Programm auf einen Tastendruck wartet, oder in irgendeinem Befehl in einer Schleife. [Ctrl][Break] kann jedoch unter den folgenden beiden Umständen nicht aktiv werden (in beiden Fällen müssen Sie Ihr System neu starten):

- Das Programm steckt in einer Schleife, in der alle Interrupts ausgeschaltet sind.
- Das System ist aufgrund eines fehlerhaften Programms abgestürzt.

Wenn Sie ein Programm debuggen, das selbst mit der [Ctrl][Break]-Kombination arbeitet, können Sie die Interrupt-Taste umdefinieren. Verwenden Sie das Installationsprogramm TDINST, um diese Taste zu verändern. Sie können als Interrupt-Taste jede beliebige Taste in Kombination mit [Ctrl] auswählen.

Programm beenden

Wenn Ihr Programm beendet wird und zu Windows zurückkehrt, übernimmt der Debugger wieder die Kontrolle. Er zeigt Ihnen den Abschlußcode, mit dem Ihr Programm zu Windows zurückkehrt. Nachdem Ihr Programm beendet wurde, können Sie die Optionen des *Run*-Menüs nur dann wieder verwenden, falls Sie Ihr Programm mit *Run / Program reset* erneut laden.

Die Segmentregister und der Stack sind normalerweise nicht mehr in Ordnung, wenn Ihr Programm beendet wurde. Sie sollten also nach dem Programmende keine Programmvariablen mehr untersuchen oder verändern.

Eine Debuggersitzung erneut starten

Turbo Debugger bietet einige Möglichkeiten, mit deren Hilfe man eine Debugger-Sitzung so einfach wie möglich erneut starten kann. Beim Debuggen passiert es leicht, daß man etwas zu weit geht und über das Ziel hinausschießt. Sie müssen den Debugger dann erneut starten und mit der Ausführung des Programms vor den letzten Befehlen aufhören, bei denen Sie außer Tritt gekommen sind.

Bei den meisten Debuggern sind Sie dazu gezwungen, alle Befehle noch einmal von Hand einzugeben, um zur entsprechenden Stelle im Programm zu kommen. Das kann sehr mühsam sein. Turbo Debugger hat eine sehr leistungsfähige Option, mit der Sie die Tastendrücke der letzten Sitzung aufzeichnen und bei Bedarf wieder abspielen können. Man kann damit auch wieder das letzte Programm mit den entsprechenden Kommandozeilenoptionen laden.

Programm erneut laden

Um das Programm wieder zu laden, das Sie gerade bearbeitet haben, wählen Sie den Befehl *Run/Program reset* aus. Turbo Debugger lädt dann das Programm mit allen Daten, die Sie seit der letzten Abspeicherung hinzugefügt haben. Das ist der sicherste Weg, um ein Programm erneut zu starten. Einfach an den Anfang des Programms zu springen, ist sehr riskant, da viele Programme davon

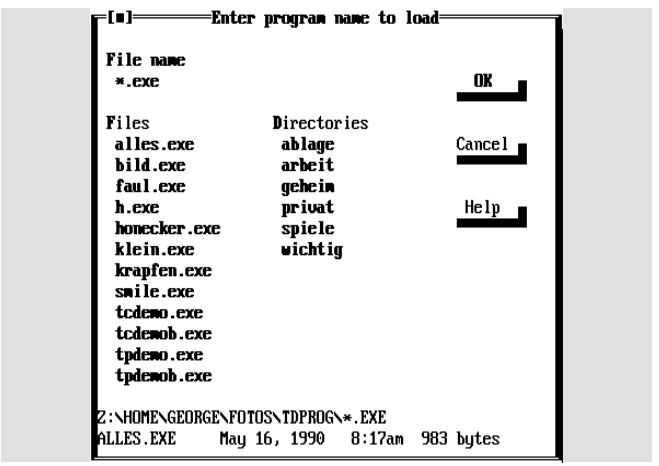


Abbildung 5.5 Dialogfenster Load Program zum Laden von Programmen

ausgehen, daß bestimmte Daten bereits in der Programmdatei mit einem bestimmten Wert belegt sind. Beachten Sie, daß beim Befehl *Program reset* alle Breakpoints und *Watches*-Ausdrücke erhalten bleiben.

Ein neues Programm zum Debuggen laden

Mit Hilfe des Befehls *File/Open* können Sie ein neues Programm zum Debuggen laden. Sie können einen Dateinamen (mit der Erweiterung ».EXE«) im Eingabefeld eingeben oder ↵ drücken, um eine Liste aller ».EXE«-Dateien des aktuellen Verzeichnisses zu erhalten. Gehen Sie mit dem Auswahlbalken zur Datei, die Sie laden wollen und drücken Sie ↵.

Wenn Sie stattdessen den Namen der gewünschten Datei eingeben, bewegt sich der Auswahlbalken zu der Datei, die mit dem/den ersten eingebenen Buchstaben beginnt. Wenn der Auswahlbalken auf der gewünschten Datei steht, drücken Sie ↵

Sie können zusätzliche Argumente für das Programm, das Sie debuggen wollen, hinter dem Programmnamen eingeben:

```
myprog a b c
```

Damit wird das Programm `MyProgy` mit drei Kommandozeilenoptionen, nämlich `a`, `b` und `c` geladen.

Die Argumente eines Programms verändern

Wenn Sie beim Laden Ihres Programms ein wichtiges Argument vergessen haben, dann können Sie den Befehl *Run/Arguments* auswählen, um das fehlende Argument einzugeben oder zu verändern. Geben Sie die neuen Argumente genauso ein, wie Sie sie in der Kommandozeile hinter dem Programmnamen angeben würden.

Sobald Sie die neuen Argumente eingegeben haben, fragt Sie Turbo Debugger, ob Sie Ihr Programm noch einmal vom Laufwerk laden wollen. Sie sollten normalerweise immer *Yes* (engl. ja) eingeben, da die Argumente in den meisten Programmen erst nach dem Laden des Programms verfügbar sind.

6. Programmdaten untersuchen und ändern

Turbo Debugger bietet einmalige Möglichkeiten, um intuitiv auf die Daten eines Programms zuzugreifen.

- Mit *Inspector*-Anweisungen können Sie Daten in der Art und Weise untersuchen, wie sie in Ihrer Quelldatei auftreten. Sie können Zeigern »folgen«, Arrays durchsuchen und Strukturen, Records und Varianten so betrachten, wie Sie sie in Ihre Quelldatei geschrieben haben.
- Variablen und Ausdrücke können in *Watches*-Fenstern untergebracht werden, wo Sie die Werte während der Programmausführung ständig kontrollieren können.
- Im Dialogfenster *Evaluate/Modify* können Sie sich den Inhalt jeder Variablen anzeigen lassen und einen neuen Wert zuweisen.

In diesem Kapitel wird davon ausgegangen, daß Sie die Datenstrukturen der Programmiersprache, die Sie verwenden (C, Pascal oder Assembler), verstehen. Wenn Sie noch nicht viel mit der Programmiersprache gearbeitet haben und noch nicht alle Datentypen kennen, dann erhalten Sie in diesem Kapitel genügend Informationen über die einfachen Datentypen (*char, integer, boolean, real* usw.). Wenn Sie sich in die komplizierteren Datentypen (Zeiger, Records, Strukturen, Varianten usw.) vertieft haben, dann sollten Sie zu diesem Kapitel zurückkehren, um mehr darüber zu erfahren, wie diese Datentypen mit Turbo Debugger betrachtet werden können.

In diesem Kapitel wird gezeigt, wie Variablen in Ihrem Programm untersucht und verändert werden können. Zunächst erläutern wir das Menü *Data* und die darin enthaltenen Befehle. Daraufhin besprechen wir, wie man Programmdaten mit Ausdrücken, die Seiteneffekte aufweisen, ändern kann. Anschließend wird dargestellt, wie man auf Datenobjekte in Ihren Quellmodulen direkt zugreifen kann. Wir stellen das *Watches*-Fenster vor und beschreiben zum Abschluß die Darstellung einfacher Datentypen für jede einzelne Sprache in einem *Inspector*-Ausschnitt.

Wenn Sie einen beliebigen Speicherbereich als Hex-Daten untersuchen oder verändern wollen, sollten Sie in Kapitel 12, »Debuggen auf Assembler-Ebene«, weiterlesen. Dort wird das Debuggen auf Assembler-Ebene dargestellt.

Das Menü Data

Im *Data*-Menü kann man Programmdaten untersuchen und verändern. Sie können dort auch Ausdrücke auswerten, die den Wert von Variablen verändern und *Inspector*-Fenster öffnen, um den Inhalt Ihrer Daten darzustellen.

Inspect...

Nach dem Befehl *Inspect* werden Sie um eine Variable gebeten, die die von Ihnen gewünschten Daten enthält. Anschließend wird ein *Inspector*-Fenster geöffnet, das den Inhalt der Variablen oder des Ausdrucks anzeigt. Sie können einen einfachen Variablennamen oder auch einen komplexen Ausdruck angeben. Der Ausdruck muß sich lediglich auf einen Speicherbereich beziehen und darf nicht einfach nur eine Konstante ergeben.

Wenn sich der Cursor zum Zeitpunkt des Befehls im Textausschnitt befindet, erhalten Sie im Dialogfenster sofort den Namen der Variablen, auf der der Cursor steht. Wenn Sie einen Ausdruck mit der Taste Ins ausgewählt haben, dann enthält die Eingabeaufforderung den selektierten Ausdruck. Die *Inspector*Fenster werden erst dann wirklich interessant, wenn es um komplizierte Datenstrukturen geht, zum Beispiel um ein Array oder um eine verkettete Liste. Da Sie die Objekte in einem *Inspector*-Fenster genau untersuchen können, können Sie ebenso leicht durch die Datenstrukturen in Ihrem Programm »wandern« wie durch Ihren Quelltext im *Module*-Fenster.

Hinweis: Sehen Sie im nachfolgenden Abschnitt, »Inspector-Fenster«, nach, welche Möglichkeiten die *Inspector*-Fenster bieten.)

Evaluate/ Modify...

Beim Befehl *Evaluate/Modify* werden Sie um einen Ausdruck gebeten, der ausgewertet werden soll. Dieser Ausdruck wird dann genauso ausgewertet wie vom Compiler. In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Erörterung der Ausdrücke.

Wenn sich der Cursor bei diesem Befehl in einem Textausschnitt befindet, so wird Ihnen die Variablenbezeichnung angeboten, auf der der Cursor gerade steht. Haben Sie einen Ausdruck mit Ins ausgewählt, wird Ihnen der markierte Ausdruck zur Übernahme angeboten.

Vergessen Sie nicht, daß Sie eine Formatangabe hinter dem Ausdruck, den Sie auswerten wollen, angeben können. Turbo Debugger stellt das Ergebnis des Datentyps dar. Um das Ergebnis in einem anderen Format darzustellen, schreiben Sie ein Komma nach dem Ausdruck und dann eine Formatangabe. In Kapitel 9, »Ausdrücke«, finden Sie eine Beschreibung der Formatsteuerung. Solche Angaben sind sehr praktisch, wenn man etwas beobachten will, aber als Format nicht die Standarddarstellung von Turbo Debugger verwendet werden soll.

Abbildung 6.1 Das Dialogfenster Evaluate/Modify

Das Dialogfenster ist in drei Eingabefelder aufgeteilt. In das obere Eingabefeld können Sie den Ausdruck eingeben, der ausgewertet soll. Dies ist das Eingabefeld *Evaluate*. Es enthält, wie andere Eingabefelder, eine History-Liste. Das mittlere Eingabefeld stellt das Ergebnis der Auswertung dar. Im unteren Eingabefeld können Sie einen neuen Ausdruck eingeben. Wenn der Ausdruck nicht verändert werden kann, steht in diesem Ausschnitt »Not available« (engl. nicht verfügbar), und Sie können den Cursor nicht in diesen Ausschnitt bewegen.

Ihre Eingaben in den Eingabefeldern *Evaluate* oder *New value* werden nach Auswahl des Befehlsfelds *Modify* ausgewertet. Zwischen den einzelnen Eingabefeldern können Sie wie in anderen Dialogfenstern mit den Tasten Tab und [Shift][Tab] umschalten. Wenn Sie in einem der Ausschnitte Esc drücken oder das Feld *Cancel* auswählen oder mit der Maus anklicken, wird das Dialogfenster entfernt.

Datenstrings, die zu lang für die Ausgabe im *Result*-Feld sind, werden rechts durch einen Pfeil abgeschnitten. Den Rest der Ausgabe können Sie durch Rollen nach rechts einblenden.

Wenn Sie ein Programm debuggen, das in objektorientiertem Pascal verfaßt wurde, können Sie im Dialogfenster *Evaluate/Modify* die Felder einer Objektinstanz oder die Elemente einer Klasseninstanz untersuchen. Sie können für die Ausgabe der Instanzen alle für Records zulässigen Formatangaben verwenden.

Falls Sie sich innerhalb einer Methode oder Elementfunktion befinden, berücksichtigt Turbo Debugger den Gültigkeitsbereich der `Self`/`this`-Parameter und verwendet automatisch die entsprechenden Formatangaben.

Mit Turbo Debugger können Sie auch eine Methode oder Elementfunktion innerhalb des Dialogfensters *Evaluate/Modify* aufrufen. Sie brauchen nur den Namen der Instanz gefolgt von einem Punkt und dahinter den Namen der Methode oder Elementfunktion sowie die gewünschten Parameter einzugeben. (Konstruktoren und Destruktoren lassen sich im *Evaluate*-Fenster jedoch nicht auswerten.)

Nehmen wir einmal an, es liegen folgende Deklarationen vor:

```
type
  Punkt = object
      X, Y          : Integer;
      Sichtbar      : Boolean;
```

```
    constructor   Init (InitX, InitY : Integer );
    destructor    Fertig; virtual;
    procedure     Einblenden; virtual;
    procedure     Ausblenden; virtual;
    procedure     Verschieben; (NeuesX, NeuesY : Integer);
  end;

  var
    EinPunkt : Punkt;
```

Sie könnten dann folgende Ausdrücke im *Evaluate*-Feld eingeben und würden die nebenstehenden Ergebnisse erhalten:

Ausdruck	Ergebnis
`EinPunkt.X`	`5($5):Integer`
`EinPunkt`	`(5,23,FALSE):Punkt`
`EinPunkt.Verschieben`	`@6F4F:00BE`
`EinPunkt.Verschieben(10,,10)`	Aufruf der Methode Verschieben
`EinPunkt.Einblenden()`	Aufruf der Methode Einblenden

Sie können das Eingabefeld *Evaluate/Modify* auch als einfachen »Taschenrechner« verwenden, indem Sie als Operanden Zahlenwerte statt Programmvariablen eingeben.

Add Watch...

Beim Befehl *Add Watch* (engl. beobachten) werden Sie um eine Anweisung oder eine Programmvariable gebeten, die in die Liste der Variablen im *Watches*-Fenster aufgenommen werden soll.

Wenn der Cursor bei diesem Befehl in einem Textausschnitt steht, dann enthält das Dialogfenster automatisch die Variable unter dem Cursor. Wenn Sie einen Ausdruck mit Ins selektieren, wird das Eingabefeld im Dialogfenster mit dem selektierten Ausdruck initialisiert.

Function Return

Mit dem Befehl *Function Return* können Sie den Wert abfragen, den die Funktion zum Zeitpunkt der Auswertung als Ergebnis zurückliefern würde. Sie sollten diesen Befehl kurz vor der Rückkehr zur aufrufenden Funktion verwenden.

Der Rückgabewert wird in einem *Inspector*-Fenster angezeigt, so daß Sie auch die Rückgabewerte von Zeigern und zusammengesetzten Datenobjekten bequem untersuchen können.

Durch diesen Befehl ist es nicht mehr nötig, in das *CPU*-Fenster umzuschalten, um den Registern den Rückgabewert zu entnehmen. Da zudem bekannt ist,

welcher Datentyp zurückgeliefert werden soll, und Wert daher korrekt formatiert ausgegeben werden kann, ist dieser Befehl viel einfacher als ein Hex-Dump anzuwenden.

Auf Datenobjekte in Quelldateien zeigen

Turbo Debugger hat einen leistungsfähigen Mechanismus, der Ihnen viel Tipparbeit beim Eingeben der Variablennamen erspart, wenn Sie eine Variable untersuchen wollen. In einem *Module*-Fenster können Sie den Cursor irgendwo auf einen Variablennamen positionieren und dann den lokalen Menübefehl *Inspect* eingeben, um ein *Inspector*-Fenster zur Anzeige des Variableninhalts zu erzeugen. Sie können auch einen Ausdruck, der untersucht werden soll, selektieren, indem Sie Ins drücken und den Ausdruck mit den Cursortasten hervorheben, bevor Sie den Befehl *Inspect* auswählen. (In Kapitel 8, »Dateien untersuchen und ändern«, finden Sie eine ausführliche Behandlung der *Module*Fenster.)

Das Fenster Watches

Im *Watche*s-Fenster kann man Variablen und Ausdrücke aus einem Programm auflisten, die man dauernd beobachten will. Sie können den Wert von einfachen Variablen, wie zum Beispiel einer Integer-Variablen, beobachten, aber auch komplexe Datenstrukturen überwachen – wie ein Array. Zudem kann man den Wert eines berechneten Ausdrucks, der sich nicht nur auf eine Speicherstelle bezieht, im Auge behalten, wie zum Beispiel x * y + 4.

Wählen Sie den Befehl *View/Watches* aus, um ein Watches-Fenster zu erzeugen. Das Fenster stellt eine Liste von Variablen oder Ausdrücken dar, deren Werte Sie überwachen wollen. Bei allen Objekten wird der Variablenname oder der Ausdruck auf der linken Seite angegeben, der Datentyp rechts davon. Die Werte zusammengesetzter Datenstrukturen (wie Arrays und Strukturen) werden in Klammern angegeben. Wenn nicht genügend Platz vorhanden ist, um den gesamten Namen oder Ausdruck anzugeben, wird er abgeschnitten.

Wenn Sie einen Ausdruck eingeben, der überwacht werden soll, können Sie ohne weiteres einen Variablennamen angeben, der noch nicht definiert ist, weil etwa die Funktion, in der die Variable deklariert ist, noch nicht aufgerufen

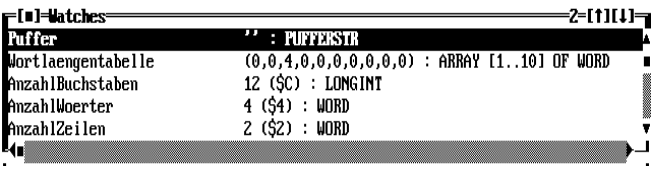


Abbildung 6.2
Das Watches-Fenster

wurde. Dadurch können Sie auch einen *Watch*-Ausdruck eingeben, bevor der Gültigkeitsbereich für den Ausdruck aktiv ist. (In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Behandlung des Gültigkeitsbereichs, und wann eine Variable oder ein Parameter definiert ist).

Achtung: Das ist die einzige Situation, in der Sie in Turbo Debugger einen Ausdruck eingeben können, der nicht sofort ausgewertet wird. Das bedeutet, daß ein Tippfehler in einem Namen nicht entdeckt wird, da Turbo Debugger davon ausgeht, daß der eingegebene Name für eine Variable steht, die erst im Laufe der Programmausführung verfügbar ist.

Falls Sie den Gültigkeitsbereich nicht, wie in Kapitel 9, »Ausdrücke«, besprochen, ausdrücklich angeben, wertet Turbo Debugger die Ausdrücke im *Watches*-Fenster nur mit dem Bezugsrahmen der aktuellen Position aus, an der Ihr Programm angehalten wurde. Daher haben die Ausdrücke in einem *Watches*-Fenster denselben Wert, den der Ausdruck in der aktuellen Position in Ihrem Programm hätte. Wenn ein *Watch*-Ausdruck einen Variablennamen enthält, der im aktuellen Bezugsrahmen nicht erreicht werden kann – da die Variable in einem anderen Modul zum Beispiel statisch deklariert wurde –, dann ist der Ausdruck nicht definiert und wird mit vier Fragezeichen (????) dargestellt.

Wenn Sie eine Objektmethode oder eine Elementfunktion einer Klasse Schritt für Schritt durchgehen, können Sie den Parameter Self/this dem *Watches*-Fenster hinzufügen.

Das lokale Menü des Watches-Fensters

Wie alle lokalen Menüs erhält man das lokale Menü des *Watches*-Fensters mit [Alt][F10]. Wenn die Ctrl-Kurzcodes aktiviert sind, kann man einen Menübefehl direkt mit Ctrl und dem ersten Buchstaben des Befehls aufrufen.

Watch...
Der Befehl *Watch* (engl. beobachten) verlangt nach einem Variablennamen oder einem Ausdruck, der in die Liste im *Watches*-Fenster eingefügt werden soll. Die Eingabe wird am Ende der Liste angehängt.

Edit...
Mit dem Befehl *Edit* kann man einen Ausdruck im *Watches*-Fenster editieren. Man kann einen vorhandenen *Watch*-Eintrag ändern oder einen neuen eingeben.

Sie können diesen Befehl auch mit der Eingabetaste ↵ oder *OK* aufrufen, wenn der Auswahlbalken über dem gewünschten *Watches*-Eintrag steht.

Remove
Das gerade ausgewählte Objekt im *Watches*-Fenster kann mit dem Befehl *Remove* (engl. entfernen) aus der *Watches*-Liste entfernt werden.

Delete all
Wenn Sie alle Einträge im *Watches*-Fenster löschen wollen, geben Sie den Befehl *Delete all* (engl. alle löschen) ein. Verwenden Sie den Befehl *Watch*, um weitere Variablen zu beobachten. Dieser Befehl wird benötigt, wenn man von einem Programmbereich in einen anderen Bereich springt, in dem die angegebenen Variablen nicht mehr von Bedeutung sind. (Anschließend können Sie mit *Watch* die anderen Variablen neu eingeben.)

Inspect
Mit dem Befehl *Inspect* (engl. untersuchen) wird ein *Inspector*-Fenster geöffnet, in dem der Inhalt des hervorgehobenen Objekts untersucht werden kann. Wenn das Objekt eine zusammengesetzte Datenstruktur hat (Array, Record oder Struktur), wird damit die Untersuchung aller Elemente ermöglicht und nicht nur derjenigen Elemente, die in das *Watches*-Fenster passen. (Der nachfolgende Abschnitt »Inspector-Fenster« enthält alle Informationen über diese Möglichkeiten.)

Change
Der Befehl *Change* (engl. verändern) erlaubt die Veränderung des gerade hervorgehobenen Objekts im *Watches*-Fenster. Im Dialogfenster können Sie den neuen Wert eingeben. Wenn die aktuell eingestellte Sprache es erlaubt, führt Turbo Debugger alle notwendigen Typumwandlungen genauso durch, als ob der passende Zuweisungsoperator (:=) verwendet worden wäre. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über den Zuweisungsoperator und die Datentypumwandlung.

Inspector-Fenster

In einem *Inspector*-Fenster werden Ihre Programmdaten entsprechend dem jeweiligen Datentyp dargestellt. Die *Inspector*-Fenster verhalten sich bei Skalaren (zum Beispiel char), Zeigern (^ in Pascal), Arrays (array[1..10] of word), Funktionen, Sets und Records unterschiedlich.

Das *Inspector*-Fenster führt alle Objekte auf, aus denen sich die Datenstruktur zusammensetzt. Die Überschrift enthält den Datentyp der untersuchten Daten und den Namen, falls einer angegeben wurde.

Der erste Eintrag in einem *Inspector*-Fenster ist immer die Speicheradresse des untersuchten Datenobjekts, die als Segment:Offset-Paar angegeben wird, sofern es sich nicht um eine Registervariable oder eine Konstante handelt.

Um den Inhalt eines *Inspector*-Fensters in Form von Bytes zu untersuchen, muß man den Befehl *View/Dump* angeben, während man sich im *Inspector*-Fenster befindet. Das *Dump*-Fenster erscheint dann mit dem entsprechenden Ausschnitt des *Inspector*-Fensters. Man kann zum *Inspector*-Fenster zurückkehren, indem man das Fenster mit *Window/Close*, [Alt][F3] oder durch Anklicken des Schließsymbols mit der Maus schließt.

In folgenden Abschnitten beschreiben wir die verschiedenen *Inspector*-Fenster, die in den jeweiligen Sprachen von Turbo Debugger unterstützt werden: Pascal und Assembler. Die verwendete Sprache bestimmt das Format der im *Inspector*-Fenster dargestellten Informationen. Datenobjekte und deren Werte erscheinen in einem Format, das der Deklaration in der Quelldatei sehr ähnlich ist.

Merken Sie sich, daß Sie nichts Spezielles unternehmen müssen, um die verschiedenen *Inspector*-Fenster zu erzeugen. Der rechte Ausschnitt erscheint automatisch so, wie es den untersuchten Daten entspricht.

Inspector-Fenster für Pascal-Daten

Skalare

Die skalaren *Inspector*-Fenster zeigen Ihnen die Werte von einfachen Datenobjekten:

```
var
  X : Integer ;
  Y : Longint ;
  Wordlen : Word ;
```

Diese *Inspector*-Fenster enthalten nur eine einzelne Zeile an Informationen. Unter der obersten Zeile wird die Adresse der Variablen angegeben. Auf der linken Seite steht der Datentyp des Skalars (byte, word, integer, longint usw.) und auf der rechten Seite steht der Wert. Der Wert kann dezimal, hexadezimal oder in beiden Formaten angegeben werden. Er wird normalerweise zuerst dezimal dargestellt, gefolgt von dem in Klammern stehenden Hexadezimalwert (mit dem Turbo-Pascal-Präfix $). Mit TDWINST können Sie die Darstellung der Werte ändern.

Wenn die dargestellte Variable den Datentyp byte oder shortint hat, wird das entsprechende ASCII-Zeichen ebenfalls dargestellt. Falls der aktuelle Wert keinem druckbaren Zeichen entspricht, wird ein Nummernzeichen (#) mit einem Hexadezimalwert verwendet. Das Nummernzeichen (#) erscheint vor den Dezimal- oder Hexadezimalwerten.

Abbildung 6.3 Inspector-Fenster mit einem Skalar in Pascal

Zeiger

Die *Inspector*-Fenster für Zeiger in einem Pascal-Programm zeigen Ihnen die Werte von Datenobjekten, die auf andere Datenobjekte zeigen. Zum Beispiel:

```
var
  IZ : ^Integer ;
  LZ : ^Pointer ;
```

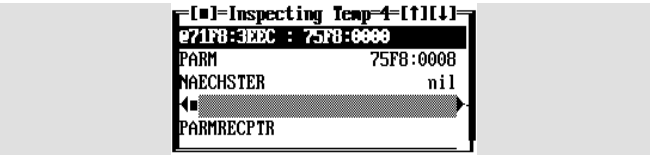


Abbildung 6.4
Ein Inspector-Fenster für einen Zeiger in Pascal

Normalerweise haben die *Inspector*-Fenster von Zeigern nur eine einzige Informationszeile, in der die Adresse der Variablen steht.

Auf der linken Seite erscheint [1], womit angedeutet wird, daß es sich um das erste Element eines Arrays handelt. Auf der rechten Seite erscheint der Wert, der vom Zeiger adressiert wird. Wenn der Wert eine komplexe Datenstruktur angibt, wie zum Beispiel ein Array oder eine Struktur, wird soviel wie möglich davon dargestellt, wobei die Werte in geschweiften Klammern, { }, stehen.

Sie können auch mehrere Zeilen darstellen, wenn Sie das Inspector-Fenster öffnen, den Befehl *Range* des lokalen Menüs wählen und eine Zahl größer 1 eingeben.

Arrays

Inspector-Fenster für Arrays zeigen die Werte der Datenobjekte eines Arrays an:

```
var
  A : array [1..10,1..20] of integer ;
  B : array[1..50] of boolean ;
```

Für jedes Element des Arrays gibt es eine Zeile. Auf der linken Seite wird der Array-Index des Elements angegeben. Auf der rechten Seite erscheint der adressierte Wert. Wenn der Wert ein komplexes Datenobjekt, wie eine Struktur oder ein Array enthält, dann werden so viele Werte wie möglich in eckigen Klammern, [und], dargestellt.

Sie können ebenfalls den Befehl *Range* des lokalen Menüs verwenden, um einen beliebigen Ausschnitt des Arrays zu untersuchen. Dies empfiehlt sich vor allem bei sehr großen Arrays, in denen man einen Ausschnitt aus der Mitte des Arrays kontrollieren will.

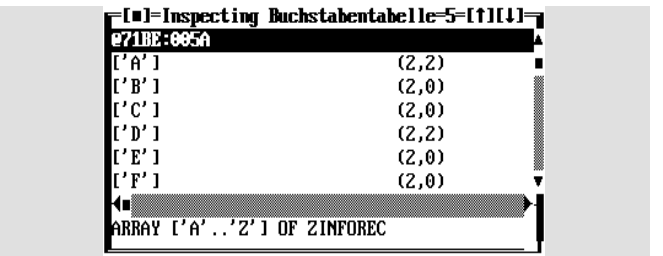


Abbildung 6.5
Ein Inspector-Fenster mit einem Array in Pascal

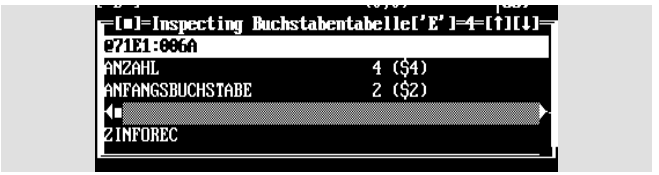


Abbildung 6.6 Ein Inspector-Fenster mit einem Pascal-Record

Records

Die *Inspector*-Fenster für Strukturen und Varianten zeigen die Werte der Elemente eines Records an:

```
Datum = record
  Jahr   : integer ;
  Monat  : 1..12 ;
  Tag    : 1..31 ;
 end ;
```

Diese *Inspector*-Fenster haben einen zweiten Ausschnitt unterhalb des ersten, in dem die Werte der Elemente angegeben sind. Dieser zusätzliche Ausschnitt zeigt den Datentyp des hervorgehobenen Elements im oberen Ausschnitt an.

Abbildung 6.7 Eine Pascal-Prozedur im Inspector-Fenster

Prozeduren und Funktionen

Im oberen Ausschnitt der *Inspector*-Fenster für Funktionen und Prozeduren erhalten Sie Informationen über die Parameter, mit denen die Routine aufgerufen wurde. Diese Fenster enthalten einen weiteren Ausschnitt, in dem die Routine als Prozedur oder Funktion gekennzeichnet und der entsprechende Datentyp angegeben wird.

Inspector-Fenster für Assemblerdaten

Skalare

Die skalaren *Inspector*-Fenster in der Assemblersprache zeigen Ihnen die Werte von einfachen Datenobjekten, wie zum Beispiel:

```
Var1     DW  99
MAGIE    DT  4.608
GROSS    DD  123456
```

Diese *Inspector*-Fenster enthalten nur eine einzelne Zeile an Informationen, die der obersten Zeile mit der Adresse der Variablen folgt. Auf der linken Seite steht der Datentyp des Skalars (`byte`, `word`, `dword`, `qword` usw.), auf der rechten Seite steht der Wert. Dieser Wert kann dezimal, hexadezimal oder in beiden

Abbildung 6.8 Skalare Assembler-Daten in einem Inspector-Fenster

Formaten angegeben werden. Er wird normalerweise dezimal dargestellt, gefolgt von dem in Klammern stehenden Hexadezimalwert (mit dem Turbo-Assembler-Suffix h). Mit TDWINST können Sie die vorgegebene Darstellung der Werte ändern.

Zeiger

Das Inspector-Fenster für Zeiger enthält in der Assemblersprache die Werte der Datenobjekte, die auf andere Datenobjekte zeigen, zum Beispiel:

```
X       DW   0
XPTR    DW   X
FARPTR  DD   X
```

Normalerweise haben die *Inspector*-Fenster von Zeigern nur eine einzige Informationszeile, in der die Adresse der Variable steht.

Auf der linken Seite steht [0], womit angedeutet wird, daß es sich um das erste Element eines Arrays handelt. Auf der rechten Seite erscheint der Wert, der von dem Zeiger adressiert wird. Wenn der Wert eine komplexe Datenstruktur angibt, zum Beispiel ein Array oder eine Struktur, wird soviel wie möglich davon dargestellt, wobei die Werte in geschweiften Klammern, { }, stehen.

Wenn der Zeiger auf einen Datentyp `byte` oder auf einen mit NUL abgeschlossenen String zeigt, erscheinen weitere Informationen. Es wird der Wert jedes Zeichens im String dargestellt. Auf der linken Seite erscheint der Array-Index ([1], [2] usw.), der Wert wird rechts angegeben, so als wäre es ein Skalar. In diesem Fall wird der gesamte String mit seiner Adresse auch noch in der Überschrift angezeigt.

Sie können auch mehrere Zeilen darstellen, wenn Sie das Inspector-Fenster öffnen, den Befehl *Range* des lokalen Menüs wählen und eine Zahl größer 1 eingeben.

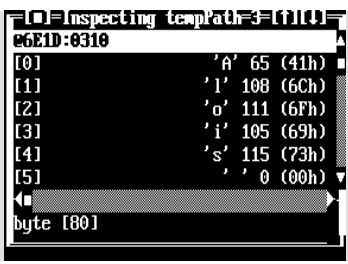


Abbildung 6.9 Ein Inspector-Fenster mit einem Zeiger in einem Assemblerprogramm

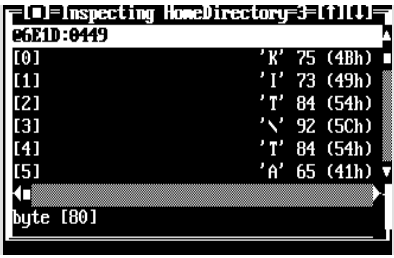


Abbildung 6.10 Ein Inspector-Fenster mit einem Array in Assembler

Arrays

In einem *Inspector*-Fenster für Arrays in Assembler werden die Werte der Datenobjekte eines Arrays angezeigt, zum Beispiel:

```
WARRAY      DW  10 DUP (0)
MELDUNG     DB  "Gruesse"
```

Jedes Element des Arrays erscheint in einer eigenen Zeile. Auf der linken Seite wird der Array-Index des Elements angegeben; auf der rechten Seite erscheint der adressierte Wert. Wenn der Wert ein komplexes Datenobjekt enthält, wie zum Beispiel eine Struktur struc, werden so viele Werte wie möglich in geschweiften Klammern, { und }, dargestellt.

Strukturen und Varianten

Die *Inspector*-Fenster für Strukturen und Varianten in Assembler zeigen die Werte der Elemente in Ihren Strukturen oder Varianten an, wie zum Beispiel von:

```
X           STRUC
Mem1        DB   ?
Mem2        DD   ?
X           ENDS
ANX         X    <1,ANX>
Y           UNION
ASBYTE      DB  10 DUP (?)
ASFLT       DT  ?
Y           ENDS

AY          Y    <?,1.0>
```

Diese *Inspector*-Fenster haben einen zweiten Ausschnitt unterhalb des ersten, in dem die Werte der Elemente angegeben sind. Dieser zusätzliche Ausschnitt zeigt den Datentyp des hervorgehobenen Elements im oberen Ausschnitt an.

Abbildung 6.16 Ein Inspector-Fenster mit einer Struktur in Assembler

Das lokale Menü des Inspector-Fensters

Die Befehle in diesem Menü machen die Leistungsfähigkeit des *Inspector*-Fensters aus. Wenn Sie den Befehl *Inspect* im lokalen Menü auswählen, können Sie weitere *Inspector*-Fenster anlegen, mit denen Sie Ihre Datenstrukturen kontrollieren können. Mit anderen Befehlen in diesem Menü kann man den Bereich der dargestellten Werte angeben und eine neue Variable untersuchen.

Um das lokale Menü des *Inspector*-Fensters zu erhalten, drücken Sie [Alt][F10]. Wenn die [Ctrl]-Tastenkürzel aktiviert sind, können Sie die Befehle direkt mit [Ctrl] und dem ersten Buchstaben des Befehls aufrufen.

Range...

Mit dem Befehl *Range* (engl. Bereich) legen Sie das erste Element und die Anzahl der Elemente fest, die dargestellt werden sollen. Verwenden Sie diesen Befehl, wenn Sie ein Array untersuchen und nur einen bestimmten Unterbereich der Elemente des Arrays sehen wollen.

Wenn Sie ein sehr großes Array haben und nur auf einige wenige Elemente in der Mitte zugreifen wollen, können Sie diesen Befehl dazu verwenden, um das *Inspector*-Fenster bei dem Array-Index zu öffnen, den Sie untersuchen wollen.

Change...

Der Befehl *Change* (engl. verändern) setzt den Wert des gerade hervorgehobenen Datenobjekts auf den Wert, welchen Sie im Dialogfenster eingeben. Wenn es die aktuelle Sprache erlaubt, führt Turbo Debugger alle notwendigen Typumwandlungen genauso durch wie der entsprechende Zuweisungsoperator. In Kapitel 9, »Ausdrücke«, erhalten Sie weitere Informationen über den Zuweisungsoperator und Typumwandlungen.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein neues *Inspector*-Fenster, in dem der Inhalt des gerade selektierten Datenobjekts dargestellt wird. Das ist sehr nützlich, wenn im *Inspector*-Fenster selbst weitere Datenobjekte enthalten sind (wie bei Strukturen eines Arrays) und Sie alle diese Objekte sehen wollen.

Wenn das aktuelle *Inspector*-Fenster eine Funktion untersucht, dann bekommen Sie mit dem Befehl *Inspect* den Quellcode der Funktion angezeigt.

Wenn der Auswahlbalken auf dem gewünschten Datenobjekt steht, können Sie diesen Befehl auch mit der Taste ↵ aktivieren.

Mit [Esc], können Sie zum vorherigen *Inspector*-Fenster zurückkehren und das *Inspector*-Fenster wieder schließen. Wenn Sie eine Datenstruktur vollständig

untersucht haben und alle *Inspector*-Fenster auf einmal entfernen wollen, können Sie den Befehl *Window/Close* oder [Alt][F3] eingeben. Mit der Maus klicken Sie dazu auf das Schließsymbol.

Descend

Der Befehl *Descend* (engl. herabsteigen) funktioniert wie der Befehl *Inspect*, mit dem Unterschied, daß ein neues Datenobjekt im aktuellen *Inspector*-Fenster eingefügt wird, statt ein neues *Inspector*-Fenster für das Datenobjekt zu öffnen. Der Befehl ist eine Kombination aus den Befehlen *New expression* und *Inspect*.

Achtung: Wenn Sie sich einmal in eine Datenstruktur begeben haben, dann können Sie zur alten Datenstruktur nicht mehr zurück. Verwenden Sie diesen Befehl, wenn Sie sich durch eine komplizierte Datenstruktur arbeiten oder eine verkettete Liste durchsuchen wollen, aber keinen Wert darauf legen, zur vorherigen Ebene zurückzukehren. Dadurch kann man die Anzahl der *Inspector*-Fenster auf dem Bildschirm sinnvoll einschränken.

New expression...

Beim Befehl *New expression* (engl. neuer Ausdruck) werden Sie um einen Variablennamen oder einen Ausdruck gebeten, der untersucht werden soll, ohne daß Sie ein weiteres *Inspector*-Fenster anlegen müssen. Damit kann man andere Daten untersuchen, ohne mehrere *Inspector*-Fenster auf dem Bildschirm halten zu müssen. Verwenden Sie diesen Befehl, wenn Sie nicht mehr an den Daten im aktuellen *Inspector*-Fenster interessiert sind.

Die *Inspector*-Fenster für Pascal-Objekte unterscheiden sich etwas von den normalen *Inspector*-Fenstern. In Kapitel 10 »Debuggen in objektorientiertem Pascal« finden Sie eine Beschreibung der *Inspector*-Fenster für Objekttypen und Klassen.

Type Cast...

Damit können Sie einen anderen Datentyp (`byte`, `word`, `char-Zeiger`, `gh2fp`, `lh2fp`) für das ausgewählte Datenelement vorgeben. Dies empfiehlt sich vor allem bei Symbolen, für die keine Datentypinformationen vorliegen, oder typenlose Zeiger. In Kapitel 11, »Debuggen unter Windows« finden Sie eine Beschreibung, wie `gh2fp` und `lh2fp` zu verwenden sind.

7. Breakpoints

Turbo Debugger verwendet den Begriff des »Breakpoints« (engl. Stoppunkt oder Haltestelle), um die drei Debuggerfunktionen zu beschreiben, die an anderer Stelle als Breakpoints, Watchpoints und Tracepoints bezeichnet werden.

Die drei Begriffe Breakpoint, Watchpoint und Tracepoint werden traditionell wie folgt definiert: Ein *Breakpoint* ist eine Stelle in Ihrem Programm, an der Sie die Programmausführung anhalten wollen, damit Sie sich Programmvariablen und Datenstrukturen genauer anschauen können. Ein *Watchpoint* dient dazu, daß Ihr Programm angehalten wird, wenn der Wert eines Ausdrucks TRUE wird. Ein *Tracepoint* bewirkt, daß Ihr Programm Zeile für Zeile ausgeführt wird, wenn sich der Wert einer bestimmten Programmvariablen oder ein Ausdruck, der sich auf eine Speicherstelle bezieht, ändert.

Turbo Debugger vereint diese drei Konzepte, indem ein Breakpoint als dreiteilige Struktur aufgefaßt wird, die aus den folgenden drei Teilen besteht:

- der Ort, an dem der Breakpoint gesetzt wird;
- die Bedingung, die für einen Breakpoint zutreffen muß;
- die Aktion, die beim Auslösen eines Breakpoints ausgeführt werden soll.

Der Ort kann entweder eine bestimmte Stelle in Ihrem Programm oder global vorgegeben sein, wobei der Breakpoint dann an jeder Stelle in Ihrem Programm auftreten kann.

Die Bedingung kann zutreffen:

- immer;
- wenn ein Ausdruck gültig wird;
- wenn sich ein Datenobjekt ändert;
- wenn eine Nachricht von Windows erscheint.

Es kann auch die Anzahl vorgegeben werden, wie oft die Bedingung zutreffen muß, bevor die Ausführung angehalten wird.

Bei der Aktion gibt es die folgenden Möglichkeiten:

- Programmausführung unterbrechen (ein Breakpoint im engeren Sinne);
- Werte von Variablen aufzeichnen;
- Ausdrücke ausführen (Code einfügen).

In diesem Kapitel zeigen wir Ihnen, wie man mit den Breakpoints in Turbo Debugger wesentlich effektiver arbeiten kann als mit den traditionellen Breakpoints, Watchpoints und Tracepoints. Sie werden das *Breakpoints*-Fenster und das *Log*-Fenster kennenlernen, werden sehen, wie man einfache Breakpoints einsetzt, Breakpoints mit Bedingungen formuliert und Breakpoints zum Protokollieren von Variablen verwendet. Schließlich werden wir zeigen, wie man einen Breakpoint setzt, um den genauen Zeitpunkt zu erfassen, an dem eine bestimmte Variable, ein Ausdruck oder ein Datenobjekt seinen Wert ändert.

Häufig werden Sie nur ein paar einfache Breakpoints setzen, damit Ihr Programm anhält, wenn es an eine bestimmte Stelle kommt. Sie können einen Breakpoint an jeder Stelle Ihres Programms setzen, indem Sie mit dem Cursor zur entsprechenden Zeile gehen und die Funktionstaste F2 drücken.

In gleicher Weise läßt sich ein Breakpoint auch im Maschinensprachecode setzen, indem man im Code-Ausschnitt des *CPU*-Fensters die Taste F2 drückt. Wenn Sie eine Maus haben, klicken Sie einfach die ersten beiden Spalten der Zeile an, in der Sie den Breakpoint setzen wollen. (Wenn Sie sich in der rechten Spalte befinden, erscheint ein ausgehöhlter Stern als Positionsanzeige.) Es gibt keine Begrenzung für die Anzahl an Breakpoints, die Sie setzen können.

Das Menü Breakpoints

Sie können das globale Menü *Breakpoints* jederzeit über die Tastenkombination [Alt][B] aufrufen.

Toggle

Mit dem Menüpunkt *Toggle* (engl. umschalten) kann man die aktuelle Zeile im *Module*-Fenster oder im Code-Ausschnitt des *CPU*-Fensters zu einem Breakpoint machen oder, falls dort bereits ein Breakpoint liegt, diesen ausschalten. Als Tastenkürzel ist die Taste F2 definiert.

At...

Unter dem Eintrag *At...* (engl. an der Stelle...) kann eine bestimmte Stelle in Ihrem Programm als Breakpoint ausgewählt werden. In dem Fenster, das sich anschließend öffnet, können Sie die Optionen für diesen Breakpoint angeben. Das Tastenkürzel lautet [Alt][F2].

Changed memory global

Mit dem Befehl *Changed memory global...* (engl. Speicherveränderung global) wird ein Breakpoint aktiviert, der dann global zur Unterbrechung der Programmausführung führt, wenn ein bestimmter Speicherbereich verändert wird. Dabei werden Sie in einem Dialogfenster gefragt, welche Speicherstelle überwacht werden soll. Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Das lokale Menü des Breakpoints-Fensters« beim Befehl *Changed memory*.

Expression true global...

Der Menüpunkt *Expression true global* (engl. Ausdruck zutreffend global) dient zur Definition von Breakpoints, die dann ausgelöst werden, wenn ein Ausdruck global gültig wird. Sie werden dann nach dem entsprechenden Ausdruck gefragt. Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Das lokale Menü des Breakpoint-Fensters« beim Befehl *Condition expression true*.

Delete all

Der Befehl *Delete All* (engl. alle löschen) entfernt alle gesetzten Breakpoints.

Der Gültigkeitsbereich von Breakpoint-Ausdrücken

Sowohl die Aktion, die beim Auslösen eines Breakpoints durchgeführt wird, als auch die Bedingung, unter der der Breakpoint ausgelöst wird, kann mit einem von Ihnen vorgegeben Ausdruck gesteuert werden. Dieser Ausdruck wird im gleichen Gültigkeitsbereich ausgewertet, in dem auch der Breakpoint gesetzt wurde, und nicht mit dem Gültigkeitsbereich, in dem das Programm eventuell unterbrochen wird. Das bedeutet, daß Ihr Breakpoint nur die Variablen verwenden kann, die zum Zeitpunkt des Setzens eines Breakpoints gültig sind, außer Sie geben den Gültigkeitsbereich ausdrücklich vor. (In Kapitel 9, »Ausdrücke«, werden die Breakpoint-Ausdrücke ausführlich beschrieben.)

Wenn Sie einen Breakpoint für einen Ausdruck in einem Modul setzen wollen, das momentan nicht geladen ist, und Turbo Debugger diesen Ausdruck nicht finden kann, können Sie entweder den Gültigkeitsbereich vorgeben, um die Datei, die den Ausrdruck enthält, zu spezifizieren oder Sie verwenden den Befehl *View/Module* um das Modul zu wechseln.

Wenn Sie lokale Variablen als Bestandteil eines Ausdrucks verwenden, wird die Überprüfung dieses Breakpoints wesentlich langsamer als bei globalen oder statischen Variablen ausgeführt.

Das Fenster Breakpoints

Das *Breakpoints*-Fenster wird durch die Auswahl des Befehls *View/Breakpoints* aufgerufen. Damit können Sie sich die einzelnen Breakpoints im Überblick anschauen und die einzelnen Bedingungen, bei deren Eintreten ein Breakpoint ausgelöst wird, anpassen. In diesem Fenster lassen sich neue Breakpoints hinzufügen, alte Breakpoints löschen und bestehende Breakpoints verändern.

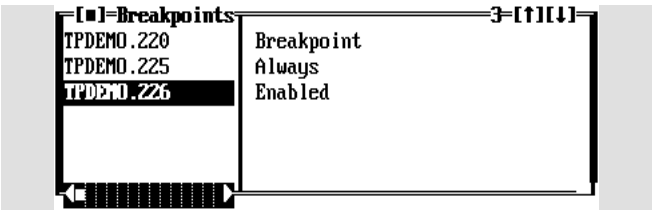


Abbildung 7.1 Das Fenster Breakpoints

Breakpoints-Fenster sind in zwei Ausschnitte unterteilt. Der linke mit der Breakpoint-Liste zeigt alle Adressen, an denen Breakpoints gesetzt wurden. Das rechte Feld mit den Breakpoint-Details enthält nähere Informationen über den aktuellen Breakpoint, der im linken Feld durch den Balkencursor markiert ist. Nur für das linke Feld gibt es ein lokales Menü, das über [Alt][F10] aufgerufen wird. Die Optionen hängen davon ab, welche Art von Breakpoint im Breakpoint-Ausschnitt ausgewählt ist.

Das lokale Menü des Breakpoint-Fensters

Mit den Befehlen in diesem Menü kann man neue Breakpoints hinzufügen, existierende Breakpoints löschen oder die Arbeitsweise eines Breakpoints verändern.

Mit [Alt][F10] erscheint das lokale Menü des *Breakpoints*-Fensters. Sind die [Ctrl]-Tastenkürzel eingeschaltet, können Sie die Taste [Ctrl] zusammen mit dem Anfangsbuchstaben des gewünschten Befehls zur direkten Ausführung des Befehls verwenden.

Set options...

Der Befehl *Set Options* öffnet das Dialogfenster *Breakpoint options*, das zwei Sätze von Auswahlschaltern, vier Eingabefelder und zwei Schaltfelder enthält. In diesem Dialogfenster können Sie

- definieren, was passieren soll, wenn der im *Breakpoints*-Ausschnitt hervorgehobene Breakpoint ausgelöst wird;
- die Bedingungen angeben, unter denen der Breakpoint ausgelöst werden soll;
- bestimmen, wie oft der Breakpoint zutreffen muß, damit er ausgelöst wird;
- einen Breakpoint ein- oder ausschalten;
- die Adresse des Breakpoints festlegen oder ändern;
- den Breakpoint global vorgeben

Das Schaltfeld unter *Actions* hat folgende Bedeutung:

Break führt zur Unterbrechung der Programmausführung, wenn der Breakpoint ausgelöst wird. Nach dem Anhalten des Programms erscheint der Turbo-Debugger-Bildschirm, und Sie können Befehle eingeben, um die Datenstrukturen in Ihrem Programm zu überprüfen.

Execute Wenn Sie die Option *Execute* (engl. ausführen) für einen Breakpoint vorgeben, wird ein von Ihnen vorgegebener Ausdruck ausgeführt. Nach der Auswahl dieses Menüpunkts werden Sie nach dem entsprechenden Ausdruck gefragt. Der Ausdruck sollte in irgendeiner Form einen Seiteneffekt enthalten, wie zum Beispiel das Setzen einer Variablen auf einen bestimmten Wert. Man kann diese Option auch als »Code-Kitt«, eine Art temporäre Programmergänzung, betrachten, da man einen Ausdruck einfügen kann, der vor dem nachfolgenden Programmteil an der aktuellen Stelle eingefügt wird, so daß man einen Bruch durch einen Programmierfehler notdürftig »kitten« kann.

Log Über die Vorgabe *Log* (engl. Protokoll) legen Sie fest, daß der Wert eines Ausdrucks beim Auslösen des Breakpoints im *Log*-Fenster aufgezeichnet werden soll. Nach der Auswahl dieses Menüpunkts werden Sie nach dem Ausdruck gefragt, dessen Wert Sie protokollieren wollen. Achten Sie bitte darauf, daß der Ausdruck keine unerwarteten Seiteneffekte hat. In Kapitel 9, »Ausdrücke«, finden Sie eine Beschreibung der Ausdrücke und möglicher Seiteneffekte.

Das Schaltfeld *Condition* hat drei Einstellungen:

Always In der Einstellung *Always* (engl. immer) müssen keine zusätzlichen Bedingungen erfüllt werden, wenn der Breakpoint ausgelöst wird.

Changed memory... Wenn die Option *Changed memory* (engl. Speicherbereich verändert) ausgewählt wurde, wird eine Variable oder ein Objekt im Speicher beobachtet. Ändert sich der betreffende Speicherbereich, wird ein Breakpoint ausgelöst. Geben Sie den Ausdruck im Eingabefeld *Condition expression* ein. Im Ausdruck

sollte das Objekt, das beobachtet werden soll, enthalten sein. Außerdem sollte die Anzahl der Objekte, die überwacht werden soll, festgelegt sein. Die Gesamtzahl der Bytes im Speicherbereich ergibt sich aus der Größe des Objekts multipliziert mit der Anzahl der Objekte. Wenn Sie zum Beispiel folgenden Pascal-Ausdruck eingegeben hätten

```
longInt(a),4
```

dann wäre der Speicherbereich, der überwacht werden soll, 16 Bytes lang, da ein longInt 4 Bytes umfaßt und Sie vier Objekte überwachen wollten.

Wenn Sie diese Bedingung mit einem globalen Breakpoint verbinden, dann wird Ihr Programm deutlich langsamer ausgeführt, da der Speicherbereich nach jeder Zeile überprüft werden muß.

Falls Sie diese Bedingung bei einem Breakpoint verwenden, der an eine bestimmte Adresse gekoppelt ist, wird das Programm überhaupt nicht verlangsamt, wie dies bei einem globalen Breakpoint der Fall wäre, und Sie können die Variable jedesmal überprüfen, wenn die entsprechende Zeile mit dem Breakpoint ausgeführt wird.

Expression true Mit der Einstellung *Expression true* (engl. Ausdruck trifft zu) wird ein Breakpoint dann ausgelöst, wenn ein Ausdruck den Wert TRUE ergibt, also zutrifft. Nach Auswahl dieser Option können Sie den entsprechenden Ausdruck im Eingabefeld *Condition Expression* eingeben.

Im Eingabefeld *Pass count...* (engl. Durchlaufzähler) wird die Anzahl der Durchläufe festgelegt, in denen eine Breakpoint-Bedingung zutreffen muß, bevor der Breakpoint ausgelöst wird. Die Anzahl, die im Befehl *Pass Count...* angegeben wurde, wird nur dann heruntergezählt, wenn die Bedingungen, die an einen bestimmten Breakpoint gestellt wurden, zutreffen. Das bedeutet, daß der Breakpoint im Falle einer Zusatzbedingung nur dann ausgelöst wird, wenn die Bedingung so oft wie angegeben zutraf.

Im Markierungsfeld *Breakpoint Disabled* (engl. Breakpoint ausschalten) kann man den Breakpoint ein- oder ausschalten. Ein ausgeschalteter Breakpoint ist so lange unsichtbar, bis er wieder eingeschaltet wird. Es scheint, als ob er gelöscht worden wäre.

Dieser Befehl ist sehr praktisch, wenn man einen sehr komplizierten Breakpoint definiert hat, den man im Moment nicht braucht, der aber später vielleicht wieder verwendet werden soll. Dieser Befehl erspart es, den Breakpoint zu löschen und dann alle Bedingungen und Aktionen wieder neu einzugeben.

Add...

Durch den Befehl *Add* (engl. hinzufügen) wird ein Dialogfenster ähnlich *Set options* geöffnet. Sie müssen eine Adresse im Eingabefeld von *Address* einfügen.

Sie können einen Breakpoint auch ganz einfach einfügen, indem Sie die Adresse eingeben, an der der Breakpoint sitzen soll. Es erscheint dann ebenfalls ein Dialogfenster, so als ob Sie den *Add*-Befehl ausgewählt hätten.

Sobald der Breakpoint eingegeben wurde, kann man die anderen lokalen Menübefehle verwenden, um die Arbeitsweise des Breakpoints zu verändern. Wenn ein Breakpoint eingefügt wird, dann hat der Durchlaufzähler den Wert 1, die Bedingung trifft immer zu, und als Aktion wird das Programm angehalten.

Remove

Der Befehl *Remove* (engl. entfernen) entfernt den gerade hervorgehobenen Breakpoint.

Delete all

Der Befehl *Delete all* (engl. alle löschen) entfernt alle Breakpoints, sowohl die globalen als auch diejenigen, die einer bestimmten Adresse zugeordnet sind. Sie müssen anschließend wieder weitere Breakpoints angeben, wenn Sie wollen, daß Ihr Programm an einem Breakpoint anhalten soll.

Inspect

Der Befehl *Inspect* (engl. untersuchen) zeigt Ihnen die Zeile im Quelltext oder den entsprechenden Assemblerbefehl für den hervorgehobenen Breakpoint. Wenn der Breakpoint an einer Adresse sitzt, die einer Zeile in Ihrem Quelltext entspricht, wird ein *Module*-Fenster geöffnet, und der Cursor steht auf der Anweisung, auf der der Breakpoint sitzt. Ansonsten wird das *CPU*-Fenster beim Auslösen des Breakpoints geöffnet und der entsprechende Befehl im Code-Ausschnitt hervorgehoben.

Sie können diesen Befehl auch dadurch aufrufen, indem Sie die Taste ⏎ einmal drücken, wenn der Breakpoint durch den Auswahlbalken in der Breakpoint-Liste hervorgehoben ist.

Das Fenster Log

Das *Log*-Fenster wird durch die Auswahl des Befehls *View/Log* geöffnet. Dieses Fenster ermöglicht ein Protokoll der wichtigsten Ereignisse, die während eines Debuggerlaufs eingetreten sind.

Das *Log*-Fenster zeigt eine Liste von Einträgen auf dem Bildschirm. Wenn mehr als 50 Zeilen in das *Log*-Fenster geschrieben wurden, werden die oberen Zeilen aus der Liste entfernt. Um die Anzahl der Zeilen vorzugeben, können Sie entweder eine Kommandozeilenoption beim Aufruf verwenden oder die Anzahl mit Hilfe des Installationsprogramms TDINST dauerhaft einstellen. Sie können das gesamte *Log*-Fenster sichern, indem Sie den Inhalt des Fensters mit Hilfe des Befehls *Open log file* fortlaufend in eine Datei schreiben.

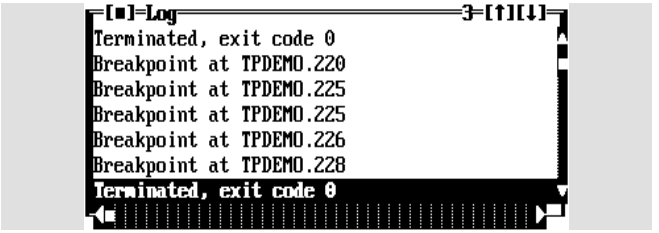


Abbildung 7.2
Das Log-Fenster

Mit folgenden Maßnahmen kann man Informationen in ein *Log*-Fenster schreiben:

- Ihr Programm hält an der von Ihnen angegebenen Stelle an. Diese Position wird in der *Log*-Datei protokolliert.
- Sie wählen den Befehl *Add comment* im lokalen Menü aus. Anschließend werden Sie nach dem Kommentar gefragt, den Sie im *Log*-Fenster aufnehmen wollen.
- Ein Breakpoint wird ausgelöst, der den Wert eines Ausdrucks in das *Log*-Fenster schreibt. Dieser Wert erscheint im Protokoll.
- Sie wählen den Befehl *Window/Dump pane to log* in der Menüleiste aus, um den aktuellen Inhalt eines Fensterausschnitts aufzuzeichnen.
- Sie debuggen eine MS-Windows-Anwendung und verwenden den Befehl *Display Windows Info* aus dem lokalen Menü des *Log*-Fensters, um Informationen über den globalen Heap, Informationen über den lokalen Heap oder die Modulliste zu protokollieren.
- Sie debuggen eine MS-Windows-Anwendung und haben mit dem Befehl *View/Windows Messages* das Fenster *Windows Messages* aufgerufen, in dessen lokalen Menü Sie sich jetzt befinden. Sie schalten *Send to Log Window* auf *Yes* um, so daß alle Meldungen, die für dieses Fenster eingehen, auch zum *Log*-Fenster übertragen werden.

Das lokale Menü des Log-Fensters

Mit den Befehlen in diesem Menü kann man das Protokoll in eine Datei schreiben, das Protokoll abbrechen und wieder aufnehmen, Kommentare einfügen und das *Log*-Fenster löschen.

Mit der Tastenkombination [Alt][F10] wird das lokale Menü des *Log*-Fensters aufgerufen. Wenn die [Ctrl]-Tastenkürzel aktiviert sind, können Sie die Befehle direkt eingeben.

Open log file...

Der Befehl *Open log file...* (engl. Log-Datei öffnen) bewirkt, daß der Text, der im *Log*-Fenster erscheint, auch in eine Datei geschrieben wird. Sie werden anschließend nach dem Namen der Datei gefragt, in die das Protokoll geschrieben werden soll. Wenn Sie eine Log-Datei öffnen, dann werden alle Zeilen,

die bereits im *Log*-Fenster dargestellt sind, in die Datei geschrieben. Daher kann eine Log-Datei auch dann noch geöffnet werden, nachdem man etwas Interessantes im *Log*-Fenster entdeckt hat, das in einer Datei gespeichert werden soll.

Wenn Sie beim Start des Dateiprotokolls die Zeilen, die bereits im *Log*-Fenster stehen, nicht in der Datei speichern wollen, dann müssen Sie vor der Auswahl von *Open log file* den Befehl *Erase log* angeben.

Close log file

Der Befehl *Close log file* (engl. Log-Datei schließen) beendet den Mitschnitt des *Log*-Fensters in der Datei, die mit dem lokalen Menübefehl *Open log file* ausgewählt wurde. Die Datei wird anschließend geschlossen.

Logging

Der Befehl *Logging* (engl. protokollieren) schaltet das Protokoll ein oder aus. Damit steuern Sie, was tatsächlich im *Log*-Fenster erscheinen soll.

Add comment

Der Befehl *Add comment* (engl. Kommentar einfügen) ermöglicht die Aufnahme von Kommentaren ins Protokoll. Sie können eine Textzeile Ihrer Wahl eingeben.

Erase log

Der Befehl *Erase log* (engl. Protokoll löschen) löscht das Protokoll. Das *Log*-Fenster ist daraufhin leer. Dieser Befehl beeinflußt aber nicht das Protokoll in der Datei.

Display Windows Info

Dieser Befehl öffnet das Dialogfenster *Windows Information*, mit dem Sie Informationen über den globalen Heap, den lokalen Heap und eine Liste der Module, aus denen Ihre Anwendung besteht, erfassen können. In Kapitel 11 »Debuggen unter Windows« erhalten Sie weitere Informationen hierüber.

Einfache Breakpoints

Ein Programm vor der Ausführung von bestimmten Codeabschnitten anzuhalten, ist eine der häufigsten Tätigkeiten beim Debuggen.

Es gibt eine Reihe von Möglichkeiten, um eine Breakpoint-Aktion festzulegen. Jede ist für bestimmte Bedingungen geeignet:

- Gehen Sie mit dem Cursor in die gewünschte Quellzeile im *Module*-Fenster und führen Sie den Befehl *Breakpoints/Toggle* aus oder drücken Sie die Taste F2 oder klicken Sie die Zeile mit Ihrer Maus an. Wenn Sie diesen Befehl in einer Zeile eingeben, in der bereits ein Breakpoint gesetzt war, wird dieser gelöscht.
- Gehen Sie mit dem Cursor zum gewünschten Befehl im Codeausschnitt des *CPU*-Fensters und geben Sie den Befehl *Breakpoints/Toggle* ein oder drücken Sie

die Taste [F2] oder klicken Sie diese Zeile mit der Maus an. Wenn Sie diesen Befehl in einer Zeile eingeben, in der bereits ein Breakpoint gesetzt war, wird dieser gelöscht.

- Führen Sie den Befehl *Breakpoints/At* aus und geben Sie die Adresse des Programmcodes ein, an dem der Breakpoint gesetzt werden soll. (Eine Adresse im Programmcode hat dasselbe Format wie ein Zeiger der aktuellen Programmiersprache. Siehe Kapitel 9, »Ausdrücke«.)
- Rufen Sie den Befehl *Add* des lokalen Menüs im Ausschnitt des *Breakpoints*-Fensters mit der Breakpoint-Liste auf und geben Sie die Adresse des Programmcodes ein, an dem der Breakpoint gesetzt werden soll.

Bedingte Breakpoints und der Durchlaufzähler

Es gibt viele Gelegenheiten, bei denen ein Breakpoint übersprungen werden soll, wenn eine bestimmte Anweisung ausgeführt wird. Das ist insbesondere dann der Fall, wenn die Zeile im Programmablauf sehr oft ausgeführt werden muß, bevor man überhaupt an der Programmstelle interessiert ist. Turbo Debugger kennt zwei Möglichkeiten, um festzulegen, wann ein Breakpoint tatsächlich ausgeführt werden soll: *Durchlaufzähler* und *Bedingungen*.

Wenn Sie Ihr Programm beim zehnten Aufruf einer Funktion anhalten wollen, können Sie einen Breakpoint am Anfang der Funktion eingeben und den Wert des Eingabefelds *Pass count* im Dialogfenster *Breakpoints* verwenden, um einzugeben, wie oft der Breakpoint übersprungen werden soll, bevor er tatsächlich ausgelöst wird.

Wenn Sie Ihr Programm nur unter einer gewissen Bedingung an einer bestimmten Stelle anhalten wollen, können Sie im Rahmen des Auswahlfelds *Expression true* einen entsprechenden Ausdruck im Dialogfenster *Breakpoint options* eingeben. Jedesmal wenn das Programm zu diesem Breakpoint kommt, wird der Ausdruck ausgewertet, und wenn die Bedingung zutrifft (Ergebnis ungleich Null), wird der Breakpoint ausgelöst. Diese Möglichkeit kann mit dem Durchlaufzähler kombiniert werden, so daß ein Breakpoint nur dann ausgelöst wird, wenn eine Bedingung so oft wie vorgegeben erfüllt war.

Sie können auch das Auswahlfeld *Changed memory* verwenden, um festzulegen, daß ein Breakpoint nur dann ausgelöst werden soll, wenn ein Datenobjekt verändert wurde. Das kann wesentlich effizienter sein, als einen globalen Breakpoint anzugeben, der genau denselben Bereich im Hinblick auf eine Veränderung des Speicherinhalts überwacht. Falls Sie etwas nur dann auf eine Veränderung hin überprüfen wollen, wenn ein bestimmter Programmabschnitt erreicht wird, kann dieser Befehl den Aufwand bei der Überwachung für Turbo Debugger enorm reduzieren.

Globale Breakpoints

Wenn Sie wollen, daß ein Breakpoint in jeder Quellzeile bzw. bei jedem Befehl ausgelöst wird, müssen Sie einen globalen Breakpoint verwenden. Es gibt mehrere Möglichkeiten, einen globalen Breakpoint zu erzeugen. Es hängt von der Situation ab, welche Methode die beste ist:

- Aktivieren Sie das Markierungsfeld *Global* im Dialogfenster *Breakpoint options*. Verwenden Sie diese Methode, wenn Sie eine Zusatzbedingung und/oder einen Durchlaufzähler angeben, oder wenn Sie eine andere Aktion beim Auslösen des Breakpoints vornehmen wollen.
- Wählen Sie den Befehl *Breakpoints/Changed memory global* aus. Verwenden Sie diesen Befehl, wenn Sie das Programm bei Veränderungen eines bestimmten Datenbereichs anhalten wollen.
- Wählen Sie den Befehl *Breakpoints/Expression true global* aus. Verwenden Sie diesen Befehl dann, wenn das Programm unter der Bedingung angehalten werden soll, daß ein Ausdruck zutrifft.

Wenn Sie einen globalen Breakpoint setzen, werden Sie in der Regel das lokale Menü des *Breakpoints*-Fensters aktivieren, um die Bedingung oder die Aktion zu modifizieren. Sonst haben Sie nichts anderes als einen Breakpoint erzeugt, der in jeder Quellzeile ausgelöst wird. Dazu hätten Sie auch den Befehl *Run/Trace* über die Menüleiste aufrufen können.

Wenn Sie einen globalen Breakpoint in jeder Quellzeile Ihres Moduls überprüfen, sollten Sie darauf achten, daß Sie nicht im *CPU*-Fenster sind, wenn Sie Ihr Programm mit einem der *Run*-Befehle von der Menüleiste aus (oder mit den entsprechenden Funktionstasten) erneut starten.

Beim Überprüfen eines globalen Breakpoints in jedem einzelnen Befehl sollten Sie darauf achten, daß das *CPU*-Fenster Ihr aktuelles Fenster ist, wenn Sie Ihr Programm mit einem der *Run*-Befehle von der Menüleiste aus (oder mit den entsprechenden Funktionstasten) erneut starten.

Achtung: Eine globale Aktion wird an jeder Quellzeile durchgeführt. Verwenden Sie einen globale Breakpoint nur dann, wenn Sie ganz genau herausfinden wollen, wann eine bestimmte Variable verändert wird oder eine Bedingung zutrifft.

Globale Breakpoints können zu einer beträchtlichen Verlangsamung Ihres Programms führen. Sie können jedoch sehr praktisch sein, wenn Ihr Programm in den Daten als Querschläger wütet.

Hinweis: Nach dem Einfügen eines globalen Breakpoints müssen Sie eine Bedingung angeben, unter der der Breakpoint ausgelöst wird.

Unterbrechung bei Datenveränderungen

Wenn Sie herausfinden wollen, in welchem Teil Ihres Programms ein bestimmter Datenbereich verändert wird, können Sie einen globalen Breakpoint mit einer der zuvor erwähnten Methoden setzen. Anschließend rufen Sie den Befehl *Changed memory* des lokalen Menüs im Detailausschnitt des Aktionsfensters auf. Geben Sie nun einen Ausdruck für den Datenbereich zusammen mit einer optionalen Anzahl an Objekten ein, die überwacht werden sollen.

Ihr Programm wird spürbar langsamer ausgeführt, wenn Sie diesen Befehl eingeben. Es ist unter Umständen sogar so langsam, daß Sie das Problem zuerst grob lokalisieren müssen, bevor Sie diese Methode anwenden können, um die Programmstelle exakt herauszufinden.

Variablenwerte protokollieren

Hin und wieder ist es praktisch, wenn man die Werte gewisser Variablen an bestimmten Programmstellen protokolliert. (Achtung: Sie können nur einen Breakpoint pro Adresse setzen.) Sie können das Ergebnis von beliebigen Ausdrücken protokollieren, zum Beispiel auch die Parameter, mit denen eine Funktion aufgerufen wird. Wenn Sie sich das Protokoll mit allen Funktionsaufrufen anschauen, können Sie leicht herausfinden, wann die Funktion mit unzulässigen Parametern aufgerufen wurde.

Wählen Sie im Dialogfenster *Breakpoint options* das Auswahlfeld *Log* aus. Daraufhin geben Sie im Eingabefeld den Ausdruck ein, dessen Wert Sie jedesmal dann protokollieren wollen, wenn der Breakpoint ausgelöst wird. Falls Sie mehrere Variablen ausgeben wollen, müssen Sie mehrere Breakpoints setzen.

Die Auswertung von Ausdrücken

Durch die Ausführung von Befehlen in Ausdrücken mit Seiteneffekten können Sie beim Auslösen eines Breakpoints neue Codeabschnitte vor bestimmten Quellzeilen einfügen. Das ist sehr praktisch, wenn man eine Routine verändert, um etwas auszuprobieren oder einen Fehler auszubessern. Bei kleineren Änderungen kann man sich damit einen Compilierungs-Link-Durchlauf sparen.

Diese Methode ist natürlich auf das Einfügen eines Ausdrucks vor der Ausführung einer bereits vorhandenen Zeile beschränkt. Zur Veränderung von Quellzeilen kann diese Methode nicht verwendet werden.

8. Dateien untersuchen und ändern

Turbo Debugger betrachtet Dateien als selbstverständliche Erweiterung des Programms, das Sie debuggen. Sie können jede beliebige Datei auf Ihrem Laufwerk untersuchen und verändern. Die Dateien können sowohl als ASCII-Text als auch als Hexadezimaldaten dargestellt werden. Sie können die Textdateien auch mit Ihrer Textverarbeitung oder Ihrem Editor von Turbo Debugger aus verändern.

In diesem Kapitel wird gezeigt, wie man zwei Arten von Dateien untersuchen und modifizieren kann: Programmquellen und die anderen Dateien. Zunächst erläutern wir, wie man Dateien mit Programmtexten untersucht und editiert, und dann zeigen wir, wie andere Dateien betrachtet und modifiziert werden können.

Programmquellen untersuchen

Die Programmquellen sind Ihre Quelldateien, die compiliert und dann zu einer ».EXE«-Datei gelinkt werden. Sie werden diese Dateien in der Regel untersuchen wollen, wenn Sie das Verhalten oder die Struktur eines bestimmten Codeabschnitts betrachten wollen. Beim Debuggen muß man oft im Quellcode einer Funktion nachschauen, weil man überprüfen will, ob die Argumente zulässig sind, oder ob die Funktion das korrekte Ergebnis zurückliefert.

Wenn Sie Ihr Programm schrittweise durchgehen, dann zeigt Turbo Debugger automatisch immer die aktuelle Position in Ihrem Programm an. Dateien, die mittels einer entsprechenden Compiler-Anweisung, wie zum Beispiel #include oder INCLUDE in Assembler oder uses in Pascal, in Ihren Programmtext aufgenommen wurden und Zeilennummern erzeugen, werden ebenfalls als Quelldateien behandelt (erscheinen also auch in der Listenauswahl von *view/Module*).

Sie sollten immer ein *Module*-Fenster geöffnet haben, damit Sie sich Ihre Programmquellen anschauen können, denn dadurch weiß Turbo Debugger, daß dieses Modul eine Quelldatei ist. Dann ist es sehr viel leichter, Breakpoints zu setzen oder Variablen zu untersuchen. Man braucht nur zur entsprechenden Stelle in der Datei zu gehen. Diese und andere Methoden werden wir im

folgenden Abschnitt beschreiben. Wie man Microsoft Windows DLLs lädt und debuggt wird in Kapitel 11 »Debuggen unter Windows« beschrieben.

Das Module-Fenster

Sie können ein *Module*-Fenster mit dem Befehl *View/Module* von der Menüleiste aus öffnen (oder Sie geben einfach das Tastenkürzel F3 ein.

Sie erhalten daraufhin im Rahmen eines Dialogfensters eine Liste mit Modulen, aus der Sie das Modul heraussuchen können, das Sie betrachten wollen.

Wenn Sie eine Windows-Anwendung debuggen, zeigt dieses Dialogfenster eine Liste für die Module des Programms und eine zweite Liste für DLLs und .EXE-Dateien, die momentan von Windows geladen sind.

Turbo Debugger wird dann das Modul laden, das Sie ausgewählt haben. Die Quelldateien werden in folgenden Verzeichnissen gesucht:

1. imVerzeichnis, in dem der Compiler die Programmdateien bei der Compilierung gefunden hat;

2. in den Verzeichnissen, die mit dem Befehl F10 /*Options/Path for source* oder der Kommandozeilenoption -sd angegeben wurden;

3. im aktuellen Verzeichnis;

4. im Verzeichnis, das das Programm enthält, das Sie debuggen.

Das *Module*-Fenster stellt den Inhalt der Quelldatei des von Ihnen ausgewählten Moduls dar. Die Überschrift des *Module*-Fensters enthält den Namen des gerade betrachteten Moduls zusammen mit dem Namen der Quelldatei und der Zeilennummer, in der der Cursor steht. Ein Pfeil in der ersten Spalte des Fensters zeigt die aktuelle Position im Programm an.

Wenn die Bezeichnung *modified* (verändert) im Titel hinter dem Dateinamen auftaucht, dann bedeutet das, daß die Quelldatei seit dem letzten Compilieren oder Linken des Programms verändert wurde. Daraus folgt unter Umständen,

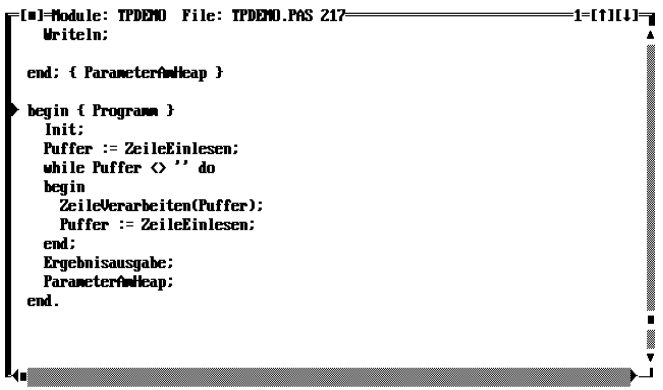


Abbildung 8.1 Das Module-Fenster

daß die Routinen in den geänderten Quelldateien nicht mehr dieselben Zeilennummern aufweisen wie die Version, die zum Erstellen des Programms, das gerade debuggt wird, verwendet wurde. Dadurch kann der Pfeil, der zur Anzeige der aktuellen Position (CS:JP) dient, in eine falsche Zeile rutschen.

Das lokale Menü des Module-Fensters

Das *Module*-Fenster enthält eine Reihe von Befehlen, mit denen man sich im dargestellten Modul bewegen, Datenobjekte untersuchen und das Fenster zur Darstellung anderer Dateien oder Module verwenden kann.

Sie werden dieses Menü wahrscheinlich öfter als jedes andere Menü in Turbo Debugger verwenden, so daß Sie mit den verschiedenen Optionen gut vertraut sein sollten.

Mit der Tastenkombination [Alt][F10] erhalten Sie das lokale Menü des *Module*-Fensters. Sind die Ctrl-Tastenkürzel aktiviert, können Sie die einzelnen Befehle auch über die Ctrl-Taste und dem Anfangsbuchstaben des gewünschten Befehls aufrufen.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein Fenster, in dem Sie den Inhalt der Programmvariablen an der aktuellen Cursorposition inspizieren können. Bevor Sie diesen Befehl aufrufen, müssen Sie den Cursor auf die Programmvariablen in der Quelldatei setzen.

Sie können auch die Taste Ins dazu verwenden, um einen Ausdruck auszuwählen, der dann hervorgehoben wird. Damit ersparen Sie sich das Eintippen eines Ausdrucks, der ohnehin im Quelltext steht.

Da Sie mit diesem Befehl nicht mehr jeden Namen einzeln eingeben müssen, werden Sie diesen Befehl wahrscheinlich sehr häufig einsetzen, um einzelne Variablenwerte zu kontrollieren.

Watch

Der Befehl *Watch* (engl. beobachten) fügt die Variable an der aktuellen Cursorposition ins *Watches*-Fenster ein. Das ist sehr praktisch, wenn man den Inhalt einer Variablen während der Programmausführung die ganze Zeit über im Auge haben will. Bevor Sie diesen Befehl aufrufen, müssen Sie den Cursor auf die zur Programmvariable in der Quelldatei setzen.

Sie können auch die Taste Ins dazu verwenden, um einen Ausdruck auszuwählen, den Sie beobachten wollen. Damit ersparen Sie sich das Eintippen eines Ausdrucks, der ohnehin im Quelltext steht.

Module...

Mit diesem Befehl kann ein anderes Modul, das aus der Liste der Module ausgewählt werden kann, in das aktuelle *Module*-Fenster geladen werden. Verwenden Sie diesen Befehl, wenn Sie am aktuellen Modul nicht mehr weiter

interessiert sind und Sie nicht mehrere Modul-Fenster auf dem Bildschirm haben wollen.

File

Der Befehl *File* ermöglicht die Darstellung einer anderen, dem aktuellen Modul zugeordneten Quelldatei. Wählen Sie die Datei, die dargestellt werden soll, aus der angebotenen Dateiliste aus. Die meisten Module bestehen nur aus einer einzigen Quelldatei. In den anderen Dateien eines Moduls werden in der Regel nur Konstanten und Datenstrukturen definiert. Sie sollten diesen Befehl verwenden, wenn sich Ihr Modul aus mehr als einer Quelldatei zusammensetzt.

Um die erste Datei anzusehen, verwenden Sie den Befehl *View/File*. Wenn Sie mehr als eine Datei gleichzeitig auf dem Bildschirm darstellen wollen, müssen Sie den Befehl *View/Another/File* über die Menüleiste auswählen, falls weitere Dateien angezeigt werden sollen.

Previous

Mit dem Befehl *Previous* (engl. vorherige Position) kommen Sie zu der Position in Ihrer Quelldatei zurück, die Sie zuletzt betrachtet haben. Sie können diesen Befehl auch dazu verwenden, um zur vorherigen Position zurückzukehren, wenn Sie einen Befehl eingegeben haben, der die Position im bearbeiteten Modul verändert hat.

Line...

Mit dem Befehl *Line* (engl. Zeile) können Sie an eine bestimmte Zeilennummer in Ihrer Datei springen. Wenn Sie eine Zeilennummer eingeben, die hinter der letzten Zeile in der Datei liegt, dann landen Sie in der letzten Zeile der Datei.

Search

Der Befehl *Search* (engl. suchen) sucht einen String ab der aktuellen Cursorposition. Wenn der Cursor an einer Position steht, an der so etwas Ähnliches wie ein Variablenname sein könnte, dann wird der Suchstring mit dieser Bezeichnung initialisiert. Haben Sie einen Block in der Datei mit der Taste Ins markiert, dann wird Ihnen dieser Block zur Übernahme angeboten, was Ihnen unnötige Tipparbeit vielleicht erspart.

Sie können auch einfache Jokerzeichen verwenden. Das Fragezeichen ? steht für ein einzelnes Zeichen und der Stern ∗ für null und mehr Zeichen. Die Suche wird nur bis zum Ende der Datei durchgeführt und fängt nicht wieder am Dateianfang an. Um die gesamte Datei zu durchsuchen, müssen Sie mit PgUp in die erste Zeile gehen.

Next

Der Befehl *Next* (engl. Nächster) sucht die nächste Stelle, in der der Suchstring, der mit dem letzten *Search*-Befehl eingegeben wurde, vorkommt. Der Befehl kann nur dann aufgerufen werden, wenn ein Suchbefehl durchgeführt wurde.

Manchmal kommt man mit einem Suchbefehl an eine völlig unerwartete Position, bevor man die Position erreicht, die man eigentlich gesucht hat. Mit *Next* kann man die Suche wiederholen, ohne den String noch einmal eingeben zu müssen.

Origin
Mit dem Befehl *Origin* (engl. Ursprung) gelangen Sie zu dem Modul und der Zeile, in der sich das Programm augenblicklich mit CS:IP befindet. Wenn das Modul, das Sie gerade dargestellt haben, nicht das Modul mit der aktuellen Programmposition ist, dann wird das *Module*-Fenster auf dieses Modul umgeschaltet. Dieser Befehl ist sehr nützlich, wenn Sie in Ihrem Programm umherstreifen wollen, um dann wieder an die aktuelle Position zurückzukehren.

Goto...
Der Befehl *Goto* (engl. gehe zu) bringt Sie an jede beliebige Stelle in Ihrem Programm. Geben Sie die Adresse an, die Sie untersuchen wollen. Sie können dazu einen Funktionsnamen oder eine Hexadezimaladresse angeben. In Kapitel 9, »Ausdrücke«, finden Sie eine vollständige Beschreibung der Möglichkeiten, eine Adresse einzugeben.
Sie können diesen Befehl auch ganz einfach aufrufen, indem Sie nur den Anfang des Labels, das Sie anschauen wollen, eingeben. Sie erhalten dann ein Dialogfenster wie beim Befehl *Goto*. Das ist eine recht praktische Kurzform für einen sehr oft verwendeten Befehl.

Andere Dateien untersuchen

Mit Hilfe des *File*-Fensters können Sie auch andere Dateien anschauen oder verändern. Die Dateien werden entweder als ASCII-Text oder in Form von Hexadezimalbytes dargestellt. Dies kann mit den Befehlen Ascii und Hex eingestellt werden, die wir in den nächsten Abschnitten besprechen.

Das File-Fenster

Mit Hilfe des Befehls kann man über die Menüleiste ein *File*-Fenster anlegen. Sie können dann die DOS-Joker verwenden, um eine Dateiliste zu erhalten, oder den genauen Namen der Datei eingeben, die geladen werden soll.
Die *File*-Fenster stellen den Inhalt der ausgewählten Datei dar. Der Name der dargestellten Datei erscheint im Falle einer ASCII-Datei zusammen mit der aktuellen Zeilennummer in der Fensterüberschrift.
Wenn Sie ein *File*-Fenster zum ersten Mal erzeugen, dann erscheint der Inhalt der Datei entweder als ASCII-Text oder in hexadezimaler Darstellung. Das hängt davon ab, ob Turbo Debugger den Inhalt der Datei als ASCII-Text oder als

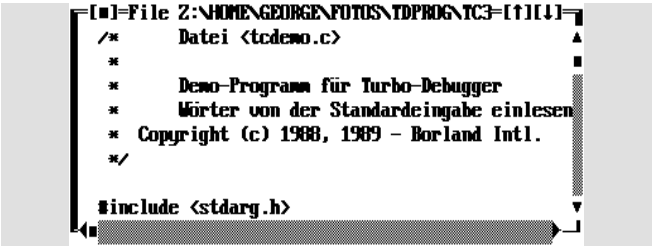


Abbildung 8.2
Das Datei-Fenster

Binärdaten interpretiert. Sie können jedoch zwischen der ASCII- und der Hexadezimaldarstellung jederzeit mit dem Befehl *Display as* umschalten, der später näher erklärt wird.

Das lokale Menü des File-Fensters

Das lokale Menü des *File*-Fensters enthält eine Reihe von Befehlen, um sich eine Datei anzusehen, die Darstellung der Datei zu verändern und die Datei selbst zu modifizieren.

Mit der Tastenkombination [Alt][F10] erhalten Sie das lokale Menü des *File*-Fensters. Sind die [Ctrl]-Tastenkürzel eingeschaltet, können Sie die einzelnen Befehle auch über die [Ctrl]-Taste zusammen mit dem Anfangsbuchstaben des gewünschten Befehls aufrufen.

Goto

Der Befehl *Goto* (engl. gehe zu) versetzt Sie direkt in eine neue Zeile oder an einen neuen Offset in der Datei. Wenn Sie die Datei als ASCII-Text ansehen, dann geben Sie die Zeilennummer ein, zu der Sie wollen. Wenn die Datei hexadezimal dargestellt wird, dann müssen Sie den Offset der gewünschten Position ab dem Dateianfang angeben. Sie können bei der Eingabe den Offset in allen möglichen Schreibweisen, die vom Parser unterstützt werden, eingeben. Wenn Sie eine Zeilennummer oder einen Offset eingeben, der hinter dem Dateiende liegt, gelangen Sie an das Ende der Datei.

Abbildung 8.3
Das Datei-Fenster mit Hexadezimal-Darstellung

Search

Der Befehl *Search* (engl. suchen) sucht ab der aktuellen Position nach einem String. Sie werden anschließend nach dem gesuchten String gefragt. Wenn der Cursor an einer Position ist, an der so etwas Ähnliches wie eine Symbolbezeichnung steht, dann wird der Suchstring mit dieser Bezeichnung initialisiert. Dasselbe passiert auch dann, wenn Sie einen Block in der Datei mit der Taste Ins markiert haben. Sie erhalten dann ebenfalls den Inhalt des Blocks als erste Vorgabe für den Suchstring. Damit können Sie sich die Tipparbeit sparen, wenn Sie ein Symbol suchen, das ohnehin bereits in der Datei steht, die Sie durchsuchen wollen. Das Format des Suchstrings hängt davon ab, ob die Datei im ASCII- oder im Hex-Modus dargestellt wird.

Wenn die Datei im ASCII-Modus dargestellt wird, dann können Sie einfache Jokerzeichen benützen. Das Fragezeichen ? einzelnes Zeichen und der Stern ∗ steht für für null und mehr Zeichen.

Wenn die Datei im Hex-Modus dargestellt wird, dann können Sie entweder eine Byte-Liste oder einen String in Anführungszeichen eingeben. Sie können den String in der Syntax angeben, die von einer der verfügbaren Sprachen unterstützt wird. In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Darstellung der Byte-Listen.

Die Suche wird nur bis zum Ende der Datei durchgeführt und fängt nicht wieder am Dateianfang an. Um die gesamte Datei zu durchsuchen, müssen Sie mit PgUp in die erste Zeile gehen.

Sie können diesen Befehl auch dadurch aktivieren, daß Sie den String, den Sie suchen, einfach direkt eintippen. Es erscheint dann sofort ein Dialogfenster, in dem Sie den Suchstring editieren können.

Next

Der Befehl *Next* (engl. Nächster) sucht die nächste Stelle des Strings, der mit dem letzten *Search*-Befehl gesucht wurde. Sie können diesen Befehl nur verwenden, nachdem Sie bereits einen *Search*-Befehl eingegeben haben.

Das ist sehr praktisch, wenn man mit dem ersten *Search*-Befehl nicht an die gewünschte Position kam. Man kann dann den Befehl so oft wiederholen, bis man dort ist, wohin man wollte.

Display as

Der Befehl *Display as* (engl. dargestellt als) schaltet die Darstellungsart der Datei um. Die Datei kann entweder als ASCII-Text oder hexadezimal dargestellt werden. Wenn Sie die ASCII-Darstellung wählen, dann erscheint die Datei so, wie Sie es von Ihrem Editor oder Ihrer Textverarbeitung gewöhnt sind. Wenn Sie die Hex-Darstellung wählen, dann steht am Anfang jeder Zeile in hexadezimaler Darstellung die Entfernung der aktuellen Position vom Dateianfang. Es werden entweder 8 oder 16 Byte in einer Zeile dargestellt. Dies hängt von der Breite des Fensterausschnitts ab. Auf der rechten Seite der *Hex*-Darstellung werden die entsprechenden ASCII-Zeichen der Byte aufgeführt. Dabei wird der vollständige Zeichensatz verwendet, so daß auch die Werte unter 32 und über 127 mit dem entsprechenden Symbol dargestellt werden.

File...

Mit dem Befehl *File...* kann man eine andere Datei im *File*-Fenster darstellen. Sie können entweder die DOS-Jokerzeichen verwenden, um eine Dateiliste zur Auswahl zu erhalten, oder den Namen der Datei, die geladen werden soll, direkt eingeben. Damit kann man eine andere Datei darstellen, ohne daß man ein neues *File*-Fenster eröffnen muß. Wenn Sie zwei verschiedene Dateien oder zwei unterschiedliche Abschnitte einer Datei gleichzeitig auf dem Bildschirm darstellen wollen, dann können Sie den Befehl *View/Another/File* angeben, um ein weiteres *File*-Fenster anzulegen.

9. Ausdrücke

Ausdrücke können eine Kombination von Symbolen Ihres Programms (also Variablen und Bezeichnungen von Routinen) sowie Konstanten und Operatoren aus einer der unterstützten Programmiersprachen C, Pascal oder Assembler sein.

Turbo Debugger kann Ausdrücke auswerten und Ihnen das Ergebnis mitteilen. Sie können Ausdrücke auch für die Bezeichnung eines Datenbereichs im Speicher verwenden, dessen Wert Sie ermitteln wollen. Sie können einen Ausdruck überall dort angeben, wo ein Wert oder eine Speicheradresse angefordert wird. (Beachten Sie, daß jede Programmiersprache einen Ausdruck unterschiedlich auswertet.)

Mit dem Befehl *Data/Evaluate/Modify* können Sie den Wert eines Ausdrucks über die Menüleiste ermitteln. Sie können diesen Befehl auch für die Abfrage eines einfachen Datenelements verwenden, sowie zum Untersuchen der Werte in Ihren Programmdaten einsetzen. Sie erhalten daraufhin ein Dialogfenster, in dem Sie nach einem Wert oder einer Speicheradresse gefragt werden.

In diesem Kapitel werden Sie lernen, wie Turbo Debugger die Programmiersprache auswählt, mit der ein Ausdruck ausgewertet wird, und wie man eine bestimmte Programmiersprache vorgeben kann. Wir beschreiben die Bestandteile der Ausdrücke, die allen Programmiersprachen gemeinsam sind, zum Beispiel die Zeilennummern in der Quelldatei und der Zugriff auf die Register des Prozessors. Anschließend werden die für jede Programmiersprache spezifischen Komponenten der Ausdrücke behandelt. Dazu zählen Konstanten, Programmvariablen, Strings und Operatoren. Für jede Sprache geben wir die Operatoren und die Syntax an, die Turbo Debugger unterstützt.

Eine komplette Behandlung der Ausdrücke in Pascal und Assembler finden Sie in Ihrem Turbo Pascal Programmierhandbuch.

Die Auswahl der Sprache für die Auswertung der Ausdrücke

Turbo Debugger bestimmt, in welcher Sprache die Ausdrücke ausgewertet werden sollen, normalerweise anhand der Dateierweiterung des aktuellen Quellmoduls. Dies ist das Modul, in dem Ihr Programm angehalten wurde. Sie können

diese Einstellung aber mit dem Befehl *Options/Language* ändern, indem Sie immer Source, C, Pascal oder Assembler im Dialogfenster vorgeben. Wenn Sie die Option *Source* auswählen, werden die Ausdrücke entsprechend dem Quellmodul ausgewertet. (Falls Turbo Debugger die Programmiersprache nicht kennt, werden die Regeln für Inline-Assembler in den Ausdrücken verwendet.)

Üblicherweise überläßt man Turbo Debugger die Auswahl der Sprache. Manchmal ist es jedoch praktisch, die Programmiersprache ausdrücklich vorzugeben, zum Beispiel beim Debuggen eines Assemblermoduls, das von einer anderen Sprache aufgerufen wird. Durch die ausdrückliche Festlegung auf eine bestimmte Programmiersprache bei der Auswertung der Ausdrücke kann man auf die Programmdaten mit dieser Sprache zugreifen, auch wenn das aktuelle Modul eine andere Sprache verwendet.

Manchmal ist es sehr praktisch, die Ausdrücke oder Variablen so zu behandeln, als wären Sie in einer anderen Sprache verfaßt. Wenn Sie zum Beispiel ein Pascal-Programm debuggen, kann es sein, daß man den Wert in einem String mit den Assembler-Konventionen leichter ändern kann.

Solange die erste Spracheinstellung beim Aufruf von Turbo Debugger in Ordnung ist, sollten Sie keine Probleme haben, andere Sprachkonventionen anzuwenden. Turbo Debugger behält die Informationen der ursprünglichen Quellsprache und verarbeitet die Umwandlungen und Speicherbefehle korrekt. Wenn ein Ausdruck mehrere Deutungen zuläßt, greift Turbo Debugger auf die Assemblersprache zurück.

Auch wenn Sie beim Aufruf von Turbo Debugger absichtlich die falsche Sprache ausgewählt haben, gibt es immer noch die Möglichkeit, einige Informationen über die ursprüngliche Quellsprache aus der Symboltabelle der Quelldatei zu entnehmen. Unter gewissen Umständen kann es jedoch sein, daß Turbo Debugger die Daten nicht richtig speichert.

Adressen im Code, Adressen von Daten, Zeilennummern

Wenn Sie auf eine Variable oder eine Funktion in Ihrem Programm zugreifen, geben Sie üblicherweise nur den entsprechenden Namen ein. Man kann jedoch auch einen Ausdruck eingeben, der einen Zeiger ergibt, oder eine Adresse im Programm als Zeilennummer angeben. Vor der Zeilennummer muß ein Nummernzeichen (#) stehen, wie zum Beispiel in #123. Im nächsten Abschnitt beschreiben wir, wie man auf Symbole zugreifen kann, die außerhalb des aktuellen Gültigkeitsbereichs liegen.

Sprache	Format	Beispiel
Pascal	$*nnnn*	$1234:$0010
Assembler	*nnnn*h	1234h:0010h
		1234h:0B234h

In Assembler muß vor Hexadezimalzahlen, die mit den Buchstaben A bis F anfangen, eine Null stehen.

Der Zugriff auf Symbole außerhalb des Gültigkeitsbereichs

Den Bereich, in dem der Debugger nach einem Symbol sucht, bezeichnen wir als aktuellen Gültigkeitsbereich dieses Symbols. Der Zugriff auf Symbole außerhalb des aktuellen Gültigkeitsbereichs ist ein erweitertes Konzept, dessen spezielle Kenntnis bei der Arbeit mit Turbo Debugger meist nicht unbedingt erforderlich ist.

Normalerweise sucht Turbo Debugger ein Symbol in einem Ausdruck auf dieselbe Art und Weise wie der Compiler.

Normalerweise sucht TDW ein Symbol in einem Ausdruck in der selben Weise, wie dies ein Compiler tun würde. In Pascal wird das Symbol zunächst in der aktuellen Prozedur oder Funktion gesucht, dann in einem der äußeren Unterprogramme (wenn das Unterprogramm in ein anderes Unterprogramm verschachtelt ist), anschließend im Implementationsabschnitt der aktuellen Unit (falls der aktuelle Gültigkeitsbereich in einer Unit liegt) und schließlich als globales Symbol.

Wenn Turbo Debugger das Symbol mit dieser Vorgehensweise nicht finden kann, werden alle anderen Module nach einem entsprechenden statischen Symbol durchsucht. Damit kann man sich auf Symbole in anderen Modulen beziehen, ohne den Modulnamen angeben zu müssen.

Wird Turbo Debugger gezwungen, das Symbol anderswo zu suchen, dann kann man die gesamte Suche exakt steuern, indem man ein Modul, eine Datei in einem Modul oder eine Routine vorgibt, in der das Symbol gesucht werden soll. Damit ist der Zugriff auf jedes Symbol, das einen definierten Wert hat, in Ihrem Programm gewährleistet, auch wenn die Symbole nur von diesen Prozeduren oder Funktionen verwendet werden dürfen und die Bezeichnungen mit anderen Symbolen verwechselt werden können.

Die Vorgabe eines Gültigkeitsbereichs

Es spielt keine Rolle, welche Programmiersprache Sie verwenden, die Methode, um den Gültigkeitsbereich für ein Symbol vorzugeben, ist immer die gleiche.

Im Regelfall verwendet man das Nummernzeichen (#), um die einzelnen Elemente eines Gültigkeitsbereichs voneinander zu trennen. Wenn es in der gerade verwendeten Programmiersprache nicht zu Verwechslungen kommt, darf man auch einen Punkt (.) anstelle des Nummernzeichens (#) verwenden.

Die folgende Syntax beschreibt, wie der Gültigkeitsbereich vorgegeben werden kann (optionale Parameter stehen in eckigen Klammern, [ ]):

```
[#Modul[#Dateiname]]#Zeilennummer[#Variable]
```

oder

```
[#Modul[#Dateiname]][#Funktion]#Variable
```

Wenn Sie kein Modul angeben, wird vom aktuellen Modul ausgegangen. Es folgen einige Beispiele für zulässige Vorgaben des Gültigkeitsbereichs in Ausdrücken mit Symbolen. Es wurde jeweils ein Beispiel einer zulässigen Kombination für die Elemente dargestellt, um den Gültigkeitsbereich vorzugeben.

Die ersten sechs Beispiele zeigen die verschiedenen Möglichkeiten, wie man Zeilennummern für die Erzeugung von Adressen und Vorgaben des Gültigkeitsbereichs verwenden kann:

`#123`	Zeile 123 im aktuellen Modul
`#123#Var1`	das Symbol `Var1` in der Zeile 123 des aktuellen Moduls
`#Mod1#123`	Zeile 123 im Modul `Mod1`
`#Mod1#123#Var1`	Symbol `Var1` in der Zeile 123 des Moduls `Mod1`
`#Mod1#Dat1#123`	Zeile 123 der Quelldatei `Dat1` des Moduls `Mod1`
`#Mod1#Dat1#123#Var1`	Symbol `Var1` in Zeile 123 der Quelldatei `Dat1` des Moduls `Mod1`

Die nächsten sechs Beispiele zeigen die verschiedenen Möglichkeiten, um den Gültigkeitsbereich einer Variaben mit Hilfe des Moduls, der Datei und der Funktionsbezeichnung vorzugeben:

`#Var2`	Variable `Var2` im aktuellen Modul
`#Funk#Var2`	Variable `Var2` in der Funktion `Funk` des aktuellen Moduls
`#Mod2#Var2`	Variable `Var2` im Modul `Mod2`
`#Mod2#Funk#Var2`	Variable `Var2` in der Funktion `Funk`, die im Modul `Mod2` definiert ist
`#Mod2#Dat2#Var2`	Variable `Var2` in der Datei `Dat2` des Moduls `Mod2`
`#Mod2#Dat2#Funk#Var2`	Variable `Var2` in der Funktion `Funk`, die in der Datei `Dat2` des Moduls `Mod2` definiert ist

Turbo Debugger unterstützt auch die Pascal-Syntax für die Vorgabe einer Unit, die das Symbol enthält:

```
Unitname.Symbol
```

Mit Hilfe von Turbo Debugger können Sie den Gültigkeitsbereich durch die Angabe eines Objekttyps, einer Klasse, einer Methode oder einer Elementfunktion aufheben. Einige Beispiele:

`Instanz`	Instanz `Instanz` des aktuellen Gültigkeitsbereichs
`Instanz.Feld`	Feld `Feld` der Instanz `Instanz`
`Objekttyp.Methode`	Methode `Methode` des Objekttyps `Objekttyp`

`Instanz.Methode`	Methode `Methode` der Instanz `Instanz`
`Junit.Instanz.Feld`	Feld Feld der Instanz `Instanz` in der Unit `Junit`
`Junit.Objekttyp.Methode`	Methode Methode des Objekttyps `Objekttyp` in der Unit `Junit`
`Junit.Objekttyp.Methode.SchachPrz.Vari`	Variable `Vari` in der verschachtelten Prozedur `SchachPrz` der Methode `Methode` des Objekttyps `Objekttyp` in der Unit `Junit`

Sie können einen Bezeichner überall dort angeben, wo auch ein Ausdruck zulässig ist – im Dialogfenster von *Evaluate/Modify*, dem *Watches*-Fenster, beim Ändern eines Ausdrucks in einem *Inspector*-Fenster oder im lokalen Menü des *Module*-Fensters bei einem Aufruf von *Goto* für eine Methode, einer Elementfunktion oder einer Prozeduradresse im Quellcode.

Der implizite Gültigkeitsbereich für die Auswertung von Ausdrücken

Jedesmal, wenn ein Ausdruck von Turbo Debugger ausgewertet wird, muß der aktuelle Gültigkeitsbereich für die Symbolbezeichnungen bestimmt werden, die keine expliziten Vorgaben für den Gültigkeitsbereich aufweisen. Das ist in vielen Programmiersprachen sehr wichtig, weil man Symbole, die denselben Namen tragen, in Funktionen oder Prozeduren wie ein globales Symbol verwenden kann. Turbo Debugger muß genau wissen, welches Symbol Sie meinen.

Normalerweise verwendet Turbo Debugger die aktuelle Cursorposition als Ausgangspunkt, um den Gültigkeitsbereich festzulegen. Sie können die Festlegung zum Beispiel vornehmen, indem Sie den Cursor auf eine bestimmten Zeile im *Module*-Fenster bewegen.

Das bedeutet, daß Sie unerwartete Ergebnisse erhalten können, wenn Sie den Cursor von der Zeile wegbewegen, an der Ihr Programm angehalten wurde. Wenn Sie sicherstellen wollen, daß die Ausdrücke im aktuellen Gültigkeitsbereich Ihres Programms ausgewertet werden, können Sie den lokalen Befehl *Origin* im *Module*-Fenster aufrufen, um wieder zur aktuellen Position in der Quelldatei zurückzukehren. Der Gültigkeitsbereich für die Ausdrücke wird auch dadurch verändert, indem man im Code-Ausschnitt des *CPU*-Fensters herumwandert, den Cursor im *Stack*-Fenster verschiebt oder im *Variables*-Fenster auf einen bestimmten Funktionsnamen stellt.

Byte-Folgen

In einigen Befehlen müssen Byte-Folgen eingegeben werden. Dazu gehören die lokalen Befehle *Search* und *Change* im Datenausschnitt des *CPU*-Fensters sowie die lokalen Befehle *Search* und *Change* im *File*-Fenster, wenn die Datei im

Hexadezimalformat dargestellt wird.Eine Byte-Folge kann eine beliebige Mischung aus skalaren Zahlen (keine Fließkommazahlen) und Strings sein, die in der Syntax der gerade ausgewählten Programmierspache angegeben werden müssen.

Die Programmiersprache kann mit dem Befehl *Options/Language* eingestellt werden. Sowohl Strings als auch die skalaren Angaben verwenden dieselbe Syntax, die auch in den Ausdrücken vorkommt. Skalare Zahlen werden in die entsprechende ByteFolge umgewandelt. Aus einem LongInt mit dem Wert 123456 in Pascal wird eine Folge von vier Bytes mit den Hexadezimalwerten H0 E2 01 00 erzeugt.

Sprache	Byte-Folge	Hex-Daten
Pascal	'ab'#4'c'	61 62 04 63
Assembler	1234 "AB"	34 12 41 42
C	"ab" 0x04 "c"	61 62 04 63

Pascal-Ausdrücke

Turbo Debugger unterstützt die Syntax der Pascal-Ausdrücke mit Ausnahme der Stringverkettung und der Mengenoperatoren. Ein Ausdruck besteht aus einer Kombination von Operatoren, Strings, Variablen und Konstanten. Im folgenden Abschnitt werden die Elemente, aus denen sich ein Ausdruck zusammensetzt, eingehend besprochen.

Pascal-Symbole

Ein Symbol ist in Pascal eine vom Benutzer definierte Bezeichnung für ein Datenobjekt oder eine Routine in einem Programm. Ein Pascal-Symbol kann mit einem Buchstaben (a-z, A-Z) oder einem Unterstrich (_) eingeleitet werden. Die nachfolgenden Zeichen in der Symbolbezeichnung können sich aus Ziffern (0-9), dem Unterstrich sowie Buchstaben zusammensetzen.

In der Regel gehorcht ein Symbol den Pascal-Regeln für den Gültigkeitsbereich eines Symbols, wobei lokal verschachtelte Symbole gegenüber anderen Symbolen mit derselben Bezeichnung den Vorrang besitzen. Sie können den Gültigkeitsbereich vorgeben, wenn Sie auf Symbole in einem anderen Gültigkeitsbereich zugreifen wollen. Weitere Details finden Sie im Abschnitt »Die Vorgabe eines Gültigkeitsbereichs«.

Konstanten und Zahlenformate in Pascal

Als Konstanten können entweder Real-Zahlen (Fließkommazahlen) oder ganze Zahlen angegeben werden. Negative Zahlen werden mit einem Minuszeichen (-) eingeleitet. Wenn die Zahl einen Dezimalpunkt oder ein e für einen Exponenten enthält, ist die Zahl eine Real-Zahl. Zum Beispiel:

```
123.4    456e34    123.45e-5
```

Ganzzahlige Konstanten werden normalerweise dezimal verarbeitet, wenn sie nicht am Anfang mit einem Dollarzeichen ($) als Hexadezimalzahl gekennzeichnet werden. Die dezimalen Integer-Konstanten müssen im Bereich von -2137483648 und 2147483647 liegen. Hexadezimale Konstanten müssen zwischen $00000000 und $FFFFFFFF liegen.

Pascal-Strings

Ein String ist eine Folge von Zeichen, die in einfachen Anführungszeichen steht, zum Beispiel

```
'abc'
```

Sie können Steuerzeichen innerhalb eines Strings angeben, wenn vor der Zahlenangabe ein Nummernzeichen (#) steht, wie in

```
'def'#7'xyz'
```

Operatoren in Pascal

Turbo Debugger unterstützt alle Pascal-Operatoren in Ausdrücken. Die einstelligen Operatoren (mit nur einem Operanden) stehen in der Reihenfolge der Auswertung an erster Stelle und haben untereinander die gleiche Priorität.

@	Adresse eines Symbols ermitteln
^	Inhalt eines Zeigers
not	Einer-Komplement
Datentyp	Datentypvorgabe
+	einstelliges Pluszeichen
-	einstelliges Minuszeichen

Die binären Operatoren (mit zwei Operanden) liegen in der Rangfolge hinter den einstelligen Operatoren. Sie sind hier in der Reihenfolge abnehmender Priorität aufgeführt:

```
*     /      div   mod   and   shl   shr
in    +      -     or    xor
<     <=     >     >=    =     <>
```

Der Zuweisungsoperator (:=) hat die niedrigste Priorität. Damit wird ebenfalls wie in C ein Wert zurückgegeben.

Der Aufruf von Funktionen und Prozeduren in Pascal

In Ausdrücken lassen sich auch Pascal-Funktionen und -Prozeduren angeben. Nehmen wir zum Beispiel an, daß Sie eine Funktion mit der Bezeichnung Halbieren deklariert haben, die einen Integer-Wert durch 2 dividiert:

```
function Halbieren ( i: integer ) : real ;
```

Sie können nun den Befehl *Data/Evaluate/Modify* auswählen und die Funktion Halbieren wie folgt aufrufen:

```
Halbieren(3)
Halbieren(10)=Halbieren(10 div 2)
```

Sie können auch Prozeduren aufrufen, aber natürlich nicht in einem Ausdruck. Wenn Sie nur einen Prozedur- oder Funktionsnamen eingeben, dann ermittelt Turbo Debugger die Adresse und die Art der Deklaration. Um eine Funktion oder eine Prozedur aufzurufen, die keine Parameter hat, muß man ein leeres Klammerpaar (( )) hinter dem Symbolnamen angeben. Zum Beispiel:

`Progy()`	Aufruf von Progy
`Progy`	Adresse und Deklaration ermitteln
`Funky=5`	Vergleich der Adresse mit dem Wert 5
`Funky()=5`	Aufruf von Funky und Vergleich des Ergebnisses mit dem Wert 5

Assembler-Ausdrücke

Turbo Debugger unterstützt die gesamte Syntax der Assemblerausdrücke. Ein Ausdruck besteht aus einer Kombination von Operatoren, Strings, Variablen und Konstanten. All diese Bestandteile eines Ausdrucks werden im folgenden Abschnitt eingehend besprochen.

Symbole in Assembler

Ein Symbol ist eine vom Benutzer definierte Bezeichnung für ein Datenobjekt oder eine Routine in einem Programm. Ein Assemblersymbol kann mit einem Buchstaben (a-z, A-Z) oder einem der vier Sonderzeichen @, ?, _ und $ anfangen. Die nachfolgenden Zeichen im Symbolnamen können sich aus Ziffern (0-9), Buchstaben und den bereits erwähnten Sonderzeichen zusammensetzen. Den Punkt (.) kann man ebenfalls als erstes Zeichen in einem Symbol verwenden, aber nicht innerhalb des Namens.

Das Spezialsymbol $ bezieht sich auf die aktuelle Position im Programm, die vom Registerpaar CS:IP angezeigt wird.

Assembler-konstanten

Als Konstanten sind entweder ganze Zahlen oder Fließkommazahlen zulässig. Eine Fließkommazahl kann einen Dezimalpunkt enthalten. Es kann sowohl die Dezimal- als auch die Exponentialschreibweise verwendet werden, wie zum Beispiel in

```
1.234   4.5e+11
```

Eine Integer-Konstante wird hexadezimal interpretiert, wenn nicht eines der folgenden Formate vorgegeben ist:

Format	Zahlenbasis
*Ziffern*H	hexadezimal
*Ziffern*O	oktal
*Ziffern*Q	oktal
*Ziffern*D	dezimal
*Ziffern*B	binär

Eine Hexadezimalzahl muß immer mit einer der Ziffern von 0 bis 9 beginnen. Wenn Sie eine Zahl eingeben wollen, die mit einem der Buchstaben von A-F anfängt, müssen Sie eine Null davorschreiben.

Operatoren in Assembler

Turbo Debugger unterstützt die meisten Assembler-Operatoren, die in der folgenden Tabelle in der Reihenfolge ihrer Priorität angeordnet wurden:

```
xxx PTR (BYTE PTR...)
. (Auswahl eines Strukturelements)
: (Segmentvorgabe)
OR XOR
```

```
AND
NOT
EQ NE LT LE GT GE
+ -
* / MOD SHR SHL
+ (einstellig) - (einstellig)
OFFSET SEG
() []
```

Die Variablen können mit dem Zuweisungsoperator (=) geändert werden, wie zum Beispiel in

```
a = [BYTE PTR DS:4]
```

Die Formatsteuerung

Wenn Sie einen Ausdruck angeben, dessen Ergebnis dargestellt werden soll, gibt Turbo Debugger das Ergebnis in einem Format aus, das vom jeweiligen Datentyp im Ausdruck abhängt. Wollen Sie das Format für die Darstellung eines Ausdrucks verändern, können Sie ein Komma hinter dem Ausdruck angeben und eine Formatangabe hinter das Komma schreiben. Die Formatangabe besteht aus einem Wiederholungsfaktor und einem Formatbuchstaben. Wiederholungsfaktoren dürfen nur bei Zeigern oder Arrays angegeben werden. Beachten Sie bitte, daß eine Formatangabe mit dem falschen Datentyp nicht beachtet wird.

Buchstabe	Format
`c`	stellt ein Zeichen oder einen Stringausdruck in jedem Fall als Zeichen dar. Nomalerweise werden alle Zeichen, die nicht ausgegeben werden können, als Escape-Sequenz oder numerisch dargestellt. Mit dieser Option werden die Zeichen mit Hilfe des vollständigen IBM-Zeichensatzes ausgegeben.
`d`	stellt eine ganze Zahl als Dezimalzahl dar.
`f[#]`	stellt eine Fließkommazahl mit der angegebenen Anzahl von Ziffern dar. Wenn Sie die Anzahl der Ziffern nicht angeben, dann werden so viele Ziffern verwendet, wie benötigt werden.
`m`	stellt einen Speicherausdruck mit Hexadezimal-Bytes dar.
`md`	stellt einen Speicherausdruck mit Dezimal-Bytes dar.
`p`	stellt den Wert als Zeiger dar, wobei Segmente wenn möglich mit einer Registerbezeichnung angegeben werden. Es wird auch das Datenobjekt, auf das der Zeiger verweist, ausgegeben. Dies ist die Voreinstellung, wenn keine Formatangabe vorliegt.

Buchstabe	Format
`s`	stellt ein Array oder einen Zeiger auf ein Zeichen-Array als String in Anführungszeichen dar. Der String muß mit NUL abgeschlossen sein.
`x` oder `h`	stellt eine ganze Zahl als Hexadezimalzahl dar.

10. Debuggen in objektorientiertem Pascal

Turbo Debugger wurde erweitert, damit die Anforderungen der Neuerungen objektorientierter Programmierung in Pascal voll erfüllt werden.

Außer den Erweiterungen für die schrittweise Bearbeitung der Methoden eines Objekts oder der Elementfunktionen einer Klasse und die Untersuchung von Objekten oder Klassen im Dialogfenster *Evaluate/modify* und dem Watches-Fenster weist Turbo Debugger eine Reihe neuer Spezialfenster und lokaler Menüs auf, die speziell für Objekte und Klassen entwickelt wurden.

Das Fenster Hierarchy

Turbo Debugger hat ein spezielles Fenster für die Untersuchung von Objekt- oder Klassenhierarchien. Dieses *Hierarchy*-Fenster erhalten Sie mit dem Befehl *View/Hierarchy*.

Das *Hierarchy*-Fenster stellt Informationen über die *Typen* von Objekten oder Klassen dar, und nicht über die Instanzen. Der linke Ausschnitt enthält die einzelnen Typen von Objekten oder Klassen, die im untersuchten Modul vorkommen. Der rechte Ausschnitt zeigt alle Objekte und Klassen in Ihren Hierarchien, wobei eine Liniengrafik verwendet wird, in der am linken Rand des Ausschnitts die Ausgangsobjekte erscheinen und die davon abstammenden Objekte rechts und unterhalb davon dargestellt werden. Die Verbindungslinien zeigen die Beziehungen zwischen Vorfahren und Nachfahren. Mit Tab können Sie zwischen den beiden Fenstern umschalten.

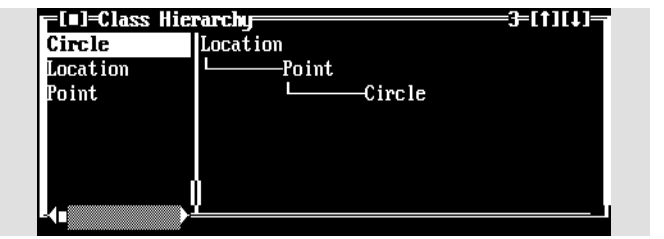


Abbildung 10.1 Das Fenster Hierarchy

Die Listenausschnitt mit den Objekttypen

Der linke Ausschnitt zeigt eine alphabetische Liste aller Objekt- oder Klassentypen, die im aktuellen Modul verwendet werden. Sie können nun mit den Pfeiltasten den Auswahlbalken auf den Objekttyp positionieren, den Sie näher untersuchen wollen. In einer sehr langen Liste von Objekttypen ist es aber einfacher, wenn Sie eine andere Vorgehensweise wählen: Beginnen Sie den Namen des gewünschten Objekttyps einzugeben. Sie werden sehen, daß der Auswahlbalken auf den Objekttyp springt, dessen Namen mit der Zeichennolge beginnt, die Sie bisher eingegeben haben. Dies bezeichnet man auch als sukzessive Auswahl.

Drücken Sie die Taste ⏎, um ein Objekttyp-/Klassen-*Inspector*-Fenster für das ausgewählte Objekt zu öffnen. Die Objekttyp-/Klassen-*Inspector*-Fenster werden nachfolgend noch ausführlich besprochen.

Das lokale Menü der Objekttyp-/Klassenliste

Drücken Sie [Alt][F10], um das lokale Menü dieses Ausschnitts anzuzeigen. Sie können auch die [Ctrl]-Tastenkürzel verwenden, wenn Sie in TDWINST aktiviert wurden. Dieses lokale Menü enthält zwei Einträge:

Inspect
öffnet ein Objekttyp-Inspector-Fenster für die ausgewählte Klasse oder den Objekttyp.

Tree
springt in den rechten Ausschnitt des Fensters, in dem der Hierarchiebaum dargestellt ist und positioniert den Auswahlbalken auf den im linken Ausschnitt ausgewählten Objekt-/Klassentyp.

Der Ausschnitt mit dem Objektbaum

Der rechte Ausschnitt stellt den Hierarchiebaum aller Objekte oder Klassen dar, die im aktuellen Modul verwendet werden. Die Beziehungen zwischen Vorfahren und Nachkommen werden durch Verbindungslinien angezeigt, wobei die Nachkommen rechts und unterhalb der Vorfahren stehen.

Um ein einzelnes Objekt in einem komplexen Hierarchiebaum zu finden, gehen Sie in den linken Ausschnitt und verwenden die oben beschriebene sukzessive Auswahl. Anschließend wählen Sie den Befehl *Tree* im lokalen Menü aus, um wieder in den Hierarchiebaum zurückzukehren. Der Auswahlbalken befindet sich dann über dem ausgewählten Objekt.

Wenn Sie ⏎ eingeben, erscheint ein Objekttyp-/Klassen-Inspector-Fenster für das ausgewählte Objekt.

Das lokale Menü des Hierarchiebaums

Das lokale Menü des Hierarchiebaums ([Alt][F10] im rechten Ausschnitt) hat nur einen Eintrag: *Inspect*. Wenn Sie diesen Befehl auswählen, erscheint ein Objekttyp-/Klassen-*Inspector*-Fenster für den ausgewählten Objekttyp. Schneller geht es jedoch, indem Sie einfach ⊟ eingeben, wenn Sie den ausgewählten Objekttyp untersuchen wollen.

Das Inspector-Fenster für Objekttypen/Klassen

Turbo Debugger hat eine neue Art von *Inspector*-Fenster, mit dem man die Einzelheiten eines Objekttyps oder einer Klasse untersuchen kann: das*Inspector*-Fenster für Objekttypen und Klassen. Dieses Fenster faßt alle Informationen über die Datentypen des Objekts zusammen, bezieht sich aber nicht auf eine bestimmte Instanz eines Objekts.

Das Fenster ist waagerecht in zwei Ausschnitte unterteilt. Der obere Ausschnitt enthält die Datenfelder des Objekttyps und der untere Ausschnitt die Bezeichnungen der Methoden und Elementfunktionen. (Falls die entsprechende Methode eine Funktion und keine Prozedur ist, wird auch der Typ des Funktionsergebnisses angezeigt.) Mit der [Tab]-Taste können Sie sich zwischen den beiden Ausschnitten dieses *Inspector*-Fensters hin- und herbewegen.

Wenn das ausgewählte Datenfeld ein Objekttyp oder ein Zeiger auf einen Objekttyp ist, können Sie mit ⊟ ein weiteres Objekttyp-/Klassen-Inspector-Fenster für den ausgewählten Typ öffnen. (Diese Aktion ist mit der Auswahl des Befehls *Inspect* im lokalen Menü dieses Fensters identisch.) Auf diese Weise lassen sich auch kompliziert verschachtelte Strukturen von Objekten mit minimalem Tippaufwand schnell untersuchen.

Damit die Darstellung übersichtlich bleibt, werden die Parameter der Methoden oder Elementfunktionen im Objekttyp-/Klassen-*Inspector*-Fenster nicht gezeigt. Um die Parameter zu untersuchen, wählen Sie die Methode oder die Elementfunktion mit ⊟ aus. Es erscheint ein *Inspector*-Fenster zur Untersuchung von Methoden und Elementfunktionen. Der obere Ausschnitt des Fensters enthält die Adresse der Implementierung der Methode oder Klasse sowie die

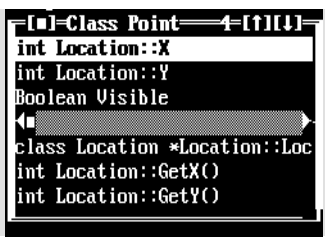


Abbildung 10.2 Das Inspector-Fenster für Objekttypen/ Klassen

Namen und Datentypen aller Parameter. Für Pascal gibt der untere Ausschnitt des Fensters an, ob die Methode eine Prozedur oder eine Funktion ist.

Drückt man irgendwo innerhalb dieses Inspector-Fensters ⊟, so erscheint das *Module*-Fenster im Vordergrund mit dem Cursor an der Stelle des Quelltexts, an der die untersuchte Methode implementiert ist.

Wie alle *Inspector*-Fenster läßt sich auch hier das aktuelle *Inspector*-Fenster mit Esc schließen; mit F3 werden alle geschlossen.

Die lokalen Menüs des Inspector-Fensters für Objekttypen/Klassen

Das lokale Menü des oberen Ausschnitts mit den Objektdaten enthält folgende Menüpunkte:

Drückt man [Alt][F10], erhält man das lokale Menü des jeweiligen Ausschnitts. Wenn die Ctrl-Tastenkürzel aktiviert sind (in TDWINST), können Sie einen lokalen Menübefehl mit Ctrl und dem ersten Buchstaben des Befehls aufrufen. Der obere Ausschnitt enthält folgende Menübefehle:

Inspect
Wenn das ausgewählte Feld ein Objekttyp, eine Klasse oder ein Zeiger auf eines dieser Elemente ist, wird ein neues *Inspector*-Fenster zur Darstellung von Objekttypen/Klassen für das ausgewählte Feld geöffnet.

Hierarchy
Der Befehl *Hierarchy* öffnet ein *Hierarchy*-Fenster für den untersuchten Objekttyp oder Klasse. Das *Hierarchy*-Fenster ist bereits zu Beginn des Kapitels beschrieben worden.

Show Inherited
Yes ist die Standardeinstellung für diesen Schalter. Wenn er auf *Yes* steht, werden alle Datenfelder und Methoden angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Felder und Methoden angezeigt, die im Typ des untersuchten Objekts definiert wurden.

Im lokalen Menü des unteren Ausschnitts sind die Befehle *Inspect*, *Hierarchy* und *Show Inherited* enthalten:

Inspect
Für das ausgewählte Element wird ein *Inspector*-Fenster für Elementfunktionen oder Methoden geöffnet. Wenn Sie zu dem Zeitpunkt [Ctrl][I] drücken, da der Cursor über der Adresse im *Inspector*-Fenster für Methoden/-Elementfunktionen steht, erscheint das *Module*-Fenster mit dem Cursor an der Stelle, an der die untersuchte Methode implementiert ist.

Hierarchy
Dieser Befehl öffnet ein *Hierarchy*-Fenster für den untersuchten Objekt- oder Klassentyp. Das *Hierarchy*-Fenster ist zu Beginn dieses Kapitels bereits beschrieben worden.

Show Inherited
Y*es* ist die Standardeinstellung dieses Schalter. Wenn er auf *Yes* steht, werden alle Methoden und Elementfunktionen angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Methoden angezeigt, die im Typ des untersuchten Objekts definiert wurden.

Inspector-Fenster für Objekt-/Klasseninstanzen

Das *Inspector*-Fenster Objekttypen und Klassen enthält Informationen über Objekttypen, sagt aber nichts darüber aus, welche Daten in einer bestimmten Instanz eines Objekts oder einer Klasse zu einem bestimmten Zeitpunkt der Programmausführung enthalten sind. Turbo Debugger hat eine erweiterte Form des altbekannten Inspector-Fensters für Records, das speziell auf die Untersuchung von Objekt- und Klasseninstanzen ausgelegt ist. Sie erhalten dieses Fenster, indem Sie den Cursor auf eine Objekt oder Klasseninstanz im *Module*-Fenster stellen und dann [Ctrl][I] drücken.

Die meisten *Inspector*-Fenster von Turbo Debugger für Records haben zwei Ausschnitte: einen oberen Ausschnitt, der die Feldbezeichnungen und deren aktuelle Werte zusammenfaßt, und einen unteren Ausschnitt, der den Datentyp des Felds angibt, das im oberen Ausschnitt ausgewählt ist. Auch ein *Inspector*-Fenster für Objekt-/Klasseninstanzen verfügt über diese beiden Ausschnitte – dazwischen ist aber noch ein dritter Ausschnitt dargestellt. Der neue Ausschnitt stellt die Methoden oder Elementfunktionen der Instanz mit ihren Programmadressen dar. (Die Programmadresse berücksichtigt auch polymorphe Objekte und die VMT.)

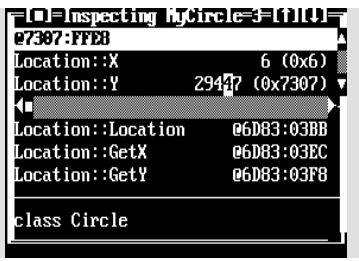


Abbildung 10.3 Das Inspector-Fenster für Objekt-/Klasseninstanzen

Die lokalen Menüs des Inspector-Fensters für Objekt- und Klasseninstanzen

Jeder der beiden oberen Ausschnitte des *Inspector*-Fensters für Objekt- und Klasseninstanzen hat sein eigenes lokales Menü, das man mit [Alt][F10] erhält. Auch hier kann man wieder die [Ctrl]-Tastenkürzel verwenden (sofern sie in TDWINST aktiviert wurden), um die einzelnen Menübefehle direkt aufzurufen.

Wie bei den *Inspector*-Fenstern von Records dient der untere Ausschnitt nur der Darstellung des Typs im ausgewählten Feld und hat kein lokales Menü. Der obere Ausschnitt, in dem die Datenfelder oder -elemente zusammengefaßt werden, weist folgende Befehle im lokalen Menü auf:

Range...
Dieser Befehl hat sich gegenüber den früheren Versionen nicht verändert. Hier wird der Bereich für die Elemente eines Arrays angegeben, der angezeigt werden soll. Wenn das untersuchte Element kein Array oder Zeiger ist, kann man diesen Befehl nicht aufrufen.

Change...
Mit diesem Befehl kann man den Wert des ausgewählten Datenfelds verändern. Dieser Befehl hat sich gegenüber den früheren Versionen von Turbo Debugger ebenfalls nicht geändert.

Methods
Dieser Befehl ist ein *Yes*/*No*-Schalter mit der Standardeinstellung *Yes*. Falls er auf *Yes* eingestellt ist, werden die Methoden oder Elementfunktionen im mittleren Ausschnitt dargestellt. Wenn er auf *No* steht, erscheint der mittlere Ausschnitt nicht. Die Einstellung wird vom nächsten geöffneten *Inspector*-Fenster übernommen.

Show Inherited
Dieser Menüpunkt funktioniert ebenfalls als *Yes*/*No*-Schalter. Wenn er auf *Yes* steht, werden alle Datenfelder und Methoden angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Methoden und Datenfelder angezeigt, die im Typ des untersuchten Objekts definiert sind.

Inspect
Wie bei den früheren Versionen von Turbo Debugger wird nach der Auswahl dieses Befehls ein *Inspector*-Fenster für die Daten des ausgewählten Felds geöffnet. Mit der Taste ↵ erreicht man dasselbe, wenn sich der Auswahlbalken über dem ausgewählten Feld oder Element befindet.

Descend
Das ausgewählte Element tritt an die Stelle des aktuellen *Inspector*-Fensters. Es wird kein neues *Inspector*-Fenster geöffnet. Sie können aber nicht mehr zum vorher untersuchten Feld zurückkehren, wie das bei der Option *Inspect* möglich

ist. *Descend* empfiehlt sich für die Untersuchung komplexer Datenstrukturen, wie Listen, da man für die einzelnen Elemente der Datenstruktur kein eigenes *Inspector*-Fenster öffnen muß.

New Expression...
Dieser Befehl fragt nach einem neuen Datenelement oder Ausdruck, der untersucht werden soll. Das neue Element ersetzt den aktuellen Ausdruck oder das aktuelle Element im Fenster. Es wird kein zusätzliches Fenster geöffnet.

Type cast...
Mit diesem Befehl können Sie einen anderen Datentyp (byte, word, integer, char, Zeiger) für das untersuchte Element vorgeben. Dieser Befehl ist vor allem dann sehr nützlich, wenn im *Inspector*-Fenster ein Symbol vorkommt, für das keine Datentypinformationen vorliegen, oder Zeiger untersucht werden soll, für den kein expliziter Typ angegeben wurde.

Hierarchy
Nach Auswahl dieses Befehls wird ein *Hierarchy*-Fenster geöffnet. Eine ausführliche Beschreibung dieses Fensters finden Sie am Anfang dieses Kapitels.

Der mittlere und der untere Ausschnitt

Im mittleren Ausschnitt werden die Methoden eines Objekts oder die Elementfunktionen einer Klasse zusammengefaßt. Der einzige Unterschied zwischen den lokalen Menüs des Methodenausschnitts und des Datenausschnitts ist das Fehlen des *Change*-Befehls. Im Gegensatz zu Datenfeldern und Datenelementen darf man die Adressen von Methoden oder Klassen während der Ausführung eines Programms nicht verändern.

Im unteren Ausschnitt wird der Datentyp des Elements dargestellt, das in den beiden oberen Fenstern ausgewählt ist.

11. Debuggen unter Windows

In diesem Kapitel werden die Funktionen des Turbo Debugger für Windows besprochen, die Ihnen Zugriff auf Windows-spezifische Informationen geben und Ihnen die Lösung der folgenden Aufgabenstellungen ermöglichen:

- Protokollieren von Botschaften, die von den Fenstern Ihres Programms gesendet und entgegengenommen werden.
- Auflistung des Inhalts des globalen Heaps.
- Auflistung des Inhalts des loaklen Heaps.
- Erstellung einer kompletten Liste der Module, die von Windows geladen werden, darin eingeschlossen auch dynamische Linkbibliotheken (DLLs).
- Debuggen von dynamischen Linkbibliotheken (DLLs).

Spezialitäten für Windows

Die folgenden Eigenschaften von TDW unterstützen das Debuggen von Windows-Programmen:

- Ein *View*-Fenster, das Fenster *Windows Messages*, das die Botschaften zeigt, die an die Fenster Ihres Programms weitergegeben werden.
- Im *Log*-Fenster können drei Arten von Daten angezeigt werden:
- Die Datensegmente im lokalen Heap Ihres Programms.
- Die Datensegmente im globalen Heap.
- Eine komplette Liste der Module (inklusive der DLLs, der dynamischen Linkbibliothen), aus denen Ihr Programm besteht.
- Typumwandlung von Speicher-Handles zu Far-Zeigern.
- Untersuchung von dynamischen Linkbibliotheken (DLLs).

Protokollieren von Windows-Botschaften

Um Botschaften zu protokollieren, die an die Fenster Ihres Programms weitergegeben werden, wählen Sie *View/Windows Messages*. Damit öffnet sich ein Fenster, das Ihnen die Botschaften zeigt, die Windows an ein oder mehrere Fenster Ihres Programms weitergibt.

Das Fenster *Windows Messages* unterteilt sich in drei Ausschnitte: dem Ausschnitt *Window Selektion* oben links, dem Ausschnitt *Message Class* oben rechts und dem Ausschnitt *Messages* unten. Die Botschaften selbst werden im Ausschnitt *Messages* angezeigt.

Auswählen eines Fensters

Bevor Sie die Botschaften protokollieren können, müssen Sie zuerst angeben, für welches Fenster Sie die Botschaften protokollieren wollen. Das tun Sie im Ausschnitt links oben, dem Ausschnitt *Window Selection*. Das lokale Menü dieses Ausschnitts wird mit [Alt][F10] aktiviert und gibt Ihnen die Möglichkeit, ein Fenster zur Protokollierung auszuwählen, zu löschen oder alle ausgewählten Fenster zu löschen.

Hinzufügen eines Fensters Um ein weiteres Fenster zur Protokollierung auszuwählen, können Sie entweder *Add* aus dem lokalen Menü des Ausschnitts wählen oder einfach damit beginnen, den Namen des Fensters in den Ausschnitt einzugeben. Beide Verfahren rufen das gleiche Dialogfenster auf.

Hinweis: Die Eingabe des ersten Werts in dieses Eingabefenster setzt die Klasse der zu prokollierenden Botschaften auf *Log all messages*.

Sie können entweder den Namen der Routine eingeben, die Meldungen für das Fenster bearbeitet (selektieren Sie den Schalter *Window Proc*), oder einen Handle-Wert eingeben (selektieren Sie den Schalter *Handle*). Geben Sie soviele Routinennamen oder Handle-Werte ein, wie notwendig sind, um die Botschaften für Ihre Fenster zu protokollieren.

Es ist einfacher, die Fenster über den Namen der Routine anzugeben, die für die Bearbeitung der Botschaften zuständig ist (zum Beispiel *WndProc*), da Sie deren Name jederzeit eingeben können, nachdem das Programm geladen wurde.

Wenn Sie es jedoch vorziehen, ein Fenster über eine Handle-Variable zu spezifizieren, müssen Sie zuerst schrittweise durch das Programm bis hinter die Zeile gehen, in der das Handle einer Variablen zugewiesen wird (verwenden Sie F7 und F8, um das Programm schrittweise ausführen zu lassen). Wenn Sie den Namen der Variablen eingeben, bevor ihr ein Handle zugewiesen wurde, kann TDW die für dieses Fenster bestimmten Botschaften nicht protokollieren.

Löschen eines Fensters Um ein ausgewähltes Fenster aus der Liste zu löschen, bewegen Sie den Cursor auf den Eintrag, rufen dann das lokale Menü (mit [Alt][F10]) auf und wählen *Remove*, oder drücken Del, [Ctrl][Y] oder [Ctrl][R].

Die gesamte Liste der ausgewählten Fenster löschen Sie mit *Delete All* aus dem lokalen Menü.

Angabe einerBotschaftsklasse

Der obere rechte Ausschnitt trägt den Namen *Message Class* und listet die zur Protokollierung ausgewählten Botschaftsklassen. Das lokale Menü dieses Ausschnitt erlaubt Ihnen ebenfalls, eine Botschaftsklasse hinzuzufügen, zu entfernen oder alle ausgewählten Botschaftsklassen zu löschen.

Sie müssen aber zuerst im Ausschnitt *Window Selection* ein Fenster angegeben haben, bevor Sie im Auschnitt *Message Class* eine Botschaftsklasse angeben können.

Wenn Sie keine spezielle Botschaft oder Botschaftsklasse zur Protokollierung angeben, protokolliert TDW alle Botschaften, die der Fenster-Prozedur oder dem Handle übergeben werden.

Hinzufügen einer Botschaftsklasse Um eine Botschaftsklasse hinzuzufügen, wählen Sie *Add* aus dem lokalen Menü des Abschnitts *Message Class*. Daraufhin erscheint das Dialogfenster *Set Message Filter*.

Im diesem Dialogfenster können Sie die zu protokollierende Meldungsklasse und die Art der Aktion angeben, die auszuführen ist, wenn die angegebene Botschaft eintrifft.

TDW protokolliert als Voreinstellung alle Botschaften, die mit *WM_* beginnen. Da viele Botschaften dieser Art ausgetauscht werden, ist es sinnvoll, die Auswahl etwas einzuschränken, um einen besseren Überblick zu erhalten. Geben Sie dazu eine der Klassen aus der Liste *Message Class* an. Es kann jeweils nur eine Klasse hinzugefügt werden. Deswegen müssen Sie, wenn Sie Botschaften mehrerer Klassen protokollieren wollen, die Option *Add* für jede dieser Klassen verwenden.

Die Meldungsklassen und deren Beschreibung:

All Messages	Alle Botschaften, die mit *WM_* beginnen.
Mouse	Botschaften, die durch Mausfunktionen erzeugt werden (zum Beispiel *WM_LBUTTONDOWN* und *WM_MOUSEMOVE*).
Window	Botschaften vom Windows-Manager (z.B. *WM_PAINT* und *WM_CREATE*).
Input	Botschaften, die über eine Tastaturfunktion oder einen Zugriff des Benutzers auf ein System-Menü, eine Bildlaufleiste oder ein Größenveränderungsfeld erzeugt werden (zum Beispiel *WM_KEYDOWN* und *WM_SYSCOMMAND*).
System	Botschaften, die durch eine systemübergreifenmde Veränderung erzeugt werden (zum Beispiel *WM_FONTCHANGE* und *WM_SPOOLERSTATUS*).
Initialization	Botschaften, die erzeugt werden, wenn eine Anwendung ein Dialog- oder ein anderes Fenster erzeugt (zum Beispiel *WM_INITDIALOG* und *WM_INITMENU*).
Clipboard	Botschaften, die erzeugt werden, wenn eine Anwendung versucht, auf die Zwischenablage eines Fensters in einer anderen

	Anwendung zuzugreifen (zum Beispiel *WM_DRAWCLIPBOARD* und *WM_SIZECLIPBOARD*).
DDE	Botschaften über den dynamischen Datenaustausch (*Dynamic Data Exchange*), die erzeugt werden, wenn zwei Anwendungen miteinader kommunizieren (zum Beispiel *WM_DDE_INITIATE* und *WM_DDE_ACK*).
Non-client	Botschaften, die von Windows erzeugt werden, um den Non-Client-Bereich eines Programmfensters einzuhalten (zum Beispiel *WM_NCHITTEST* und *WM_NCCREATE*).
Other	Alle Botschaften, die mit *WM_* beginnen, aber zu keiner der oben genannten Kategorien gehören, zum Beispiel Botschaften benutzerdefinierter Dialogelemente oder Botschaften einer Mehrfach-Dokument-Schnittstelle.
Single Message	Jede einzelne Botschaft, die mit *WM_* beginnt, und die Sie protokollieren oder bei der Sie den Programmlauf abbrechen wollen.

Um eine einzelne Botschaft zu protokollieren, wählen Sie *Single Message* und geben Sie den Namen der Botschaft oder deren Nummer ein. Bei der Eingabe eines Namens müssen Sie darauf achten, daß Sie nur Großbuchstaben verwenden.

Die voreingestellte Aktion ist die Protokollierung der Botschaft. Die andere mögliche Aktion ist der Abbruch des Programms, wenn es eine bestimmte Botschaft erhält. Das entspricht dem Setzen eines Breakpoints für eine Botschaft.

Folgendermaßen können Sie zum Beispiel die Botschaft *WM_PAINT* protokollieren und das Programm jedesmal stoppen lassen, wenn diese Botschaft zu einem Fenster gesendet wird, das Sie im Ausschnitt *Window Selection* gewählt haben:

1. Selektieren Sie den oberen rechten Ausschnitt, den Ausschnitt *Message Class*.

2. Rufen Sie das lokale Menü auf und wählen Sie *Add*.

3. Im darauf geöffneten Dialogfenster wählen Sie *Break* (im Schaltfeld *Action*) und *Single Message* (im Schaltfeld *Message Name*).

4. Geben Sie *WM_PAINT* in das Eingabefeld *Message Name* ein und drücken Sie dann Enter.

Löschen einer Botschaftsklasse

Um eine Botschaftsklasse zu löschen, bewegen Sie den Cursor auf das Element, und rufen dann entweder das lokale Menü auf und wählen *Remove* oder drücken Del oder [Ctrl][Y].

Um alle Botschaftsklassen zu löschen, wählen Sie *Delete All* aus dem lokalen Menü.

Die voreingestellte Klasse, *Log all messages*, erscheint, nachdem Sie alle Klassen gelöscht haben. Diese Klasse kann nicht mit *Remove* oder *Delete All* gelöscht werden.

Tips zu Windows-Botschaften

Wenn Sie Botschaften für mehr als ein Fenster anzeigen lassen, sollten Sie nicht alle Botschaften protokollieren, sondern vielmehr bestimmte Botschaften oder Botschaftsklassen für jedes einzelne Fenster protokollieren. Wenn Sie alle Botschaften protokollieren, kann die große Anzahl von Botschaften, die zwischen Windows und TDW ausgetauscht werden, zum Systemabsturz führen.

Wenn Sie als Aktion den Abbruch des Programms für Botschaften einstellen, die zur Botschaftsklasse *Mouse* gehören, beachten Sie, daß auf eine *MouseDown*-Botschaft immer eine *MouseUp*-Botschaft folgen muß um die Tastatur wieder zu aktivieren. Diese Einschränkung bedeutet, daß Sie, wenn Sie zur Anwendung zurückkehren, mehrmals die Maustaste klicken müssen, damit Windows die *MouseUp*-Botschaft registriert. Wenn Windows die Botschaft erhalten hat, wird dies im unteren Ausschnitt des Fensters *Windows Messages* angezeigt.

Wenn Sie zwar eine Handle-Variable eingeben, aber gleichzeitig angeben, daß es sich um eine Prozedur handelt, akzeptiert TDW Ihre Eingaben und gibt zunächst keine Fehlermeldung aus. Wenn Sie jedoch das Programm laufen lassen, protokolliert TDW keine Botschaften. Werden also keine Botschaften protokolliert, überprüfen Sie die Eingaben im Dialogfenster *Set Message Filter*, um zu sehen, ob Sie die Handle-Schalter aktiviert haben.

Anzeigen von Botschaften

Windows-Botschaften werden im unteren Ausschnitt des Fensters *Windows Messages* angezeigt. Dieser Ausschnitt kann bis zu 200 Botschaften anzeigen.

Wenn Sie die Botschaften in eine Datei speichern wollen, müssen Sie dazu eine Protokolldatei für das Protokollfenster öffnen (mit dem Menübefehlen *View/LogFile* und dann *Open log File* aus dem lokalen Menü). Schalten Sie dann zum *Message*-Ausschnitt zurück und ändern Sie die Einstellung *Send To Log Window* auf *Yes*.

Wenn Sie alle Botschaften aus dem Ausschnitt löschen wollen, wählen Sie *Erase Log* aus dem lokalen Menü. Dieser Befehl hat keine Auswirkungen auf die Botschaften, die in das *Log*-Fenster geschrieben wurden.

Erstellen von Speicher- und Modullisten

Um den Inhalt des globalen und lokalen Heaps oder der Module Ihres Windows-Programms aufzulisten, rufen Sie zuerst das *Log*-Fenster mit *View/Log* und dann dessen lokales Menü auf. Der letzte Befehl dieses Menü ist *Display Windows Info*. Wenn Sie diesen Befehl wählen, wird das Dialogfenster *Windows Information* aufgerufen. Darin können Sie die Art der Liste angeben und festlegen, an welchem Punkt die Aufzeichnung der Liste beginnen soll.

Wenn Sie *Global Heap* wählen, können Sie festlegen, ob die Liste von oben, von unten oder von der mit einem Start-Handle angegebenen Position angezeigt wird.

Ein Start-Handle ist der Bezeichner einer Variablen, der über eine Speicherzuweisungsroutine von Windows (zum Beispiel *GlobalAlloc*) ein Handle für den globalen Heap zugewiesen wurde. Die Auswahl eines Start-Handle bewirkt, daß TDW das über dieses Handle referenzierte Datenobjekt sowie die vier folgenden Objekte auf dem Heap anzeigt.

Auflistung des Inhalts des globalen Heaps

Der globale Heap ist der globale Speicherbereich, den Windows allen Programmen zur Verfügung stellt. Wenn Sie Ressourcen wie Symbole, Bitmaps, Dialogfenster oder Fonts angeben, oder wenn Sie Speicher mit der Funktion *GlobalAlloc* belegen, verwendet Ihr Programm den globalen Heap.

Um eine Liste der Datenobjekte im globalen Heap anzeigen zu lassen, wählen Sie den Schalter *Global Heap* im Dialogfenster *Windows Information* und dann *OK*. Die Datenobjekte werden im *Log*-Fenster aufgelistet.

Hinweis: Da diese Liste wahrscheinlich die Anzahl der Zeilen überschreitet, die das *Log*-Fenster aufnehmen kann, (in der Voreinstellung sind das 50 Zeilen), sollten die das Protokoll entweder in eine *Log*-Datei schreiben (über das lokale Menü *Log Window*) oder die Anzahl der Zeilen erhöhen, die das *Log*-Fenster verwenden kann (mit TDWINST). Maximal können 200 Zeilen eingestellt werden.

Das folgende Beispiel zeigt zwei Zeilen einer Auflistung des globalen Heaps gefolgt von einer Tabelle mit der Erklärung jedes Felds:

```
0EC5        00000040 PDB (0F1D)
053E (053D) 00002DC0b GDI DATA MOVEABLE LOCKED=00001 PGLOSCKED=0001
```

Feld	Beschreibung
0EC5 053E	Entweder ein Handle für das Speicherobjekt, ausgedrückt als vierstelliger Hex-Wert oder das Wort FREE, womit angegeben wird, daß der Speicherblock nicht belegt ist.
(053D)	Ein Speicherselektor, der auf einen Eintrag in der globalen Deskriptortabelle zeigt. Der Selektor wird nicht angezeigt, wenn er denselben Wert hat wie das Speicher-Handle.
00000040b 00002DC0b	Eine hexadezimale Zahl, die die Länge des Segments in Bytes angibt.
PDB	Der Allokator des Segments, normalerweise ein Programm- oder Bibiliothek-Modul. Ein PDB ist ein Prozeßdeskriptorblock, auch Programm Segment Prefix (PSP) genannt.
(0F1D)	Ein Handle, die den Besitzer der PDB angibt.
DATA	Der Typ des Speicherobjekts. Es gibt folgende Typen: DATA Datensegment eines Programms oder einer DLL CODE Code-Segment eines Programms oder einer DLL PRIV Entweder ein Systemobjekt oder globale Daten für ein Programm oder eine DLL.
MOVEABLE	Ein Speicherallokationsattribut. Ein Objekt kann FIXED, MOVEABLE oder MOVEABLE DISCARDABLE sein.

Feld	Beschreibung
LOCKED=00001	Gibt bei Objekten mit dem Attribut *moveable* oder *moveable-discardable* die Anzahl der Sperren auf dem Objekt an. Diese Sperren wurden entweder von der Funktion *GlobalLock* oder der Funktion *LockData* gesetzt.
PGLOCKED=0001	Im erweiterten 386er-Modus wird der Speicher in *Pages* von jeweils 4 Kbyte Größe verwaltet. Dieser Wert gibt an, wieviele Sperren auf den *Pages* des Objekts liegen. Gesetzt wird eine solche Sperre von der Funktion *GlobalPageLock*. Liegt eine solche Sperre auf dem Objekt, kann Windows keine der *Pages* dieses Objekts auf die Festplatte auslagern.

Auflisten der Inhalts des lokalen Heaps

Der lokale Heap ist der »private« Speicherbereich eines Programms. Er ist für andere Windows-Programme nicht verfügbar. Dies gilt auch für den Fall, daß das Programm mehrmals geladen wird.

Ein Programm muß nicht unbedingt einen lokalen Heap haben. Windows erzeugt einen lokalen Heap, wenn das Programm die Funktion *LocalAlloc* verwendet.

Um eine Liste der Datenobjekte im lokalen Heap anzeigen zu lassen, wählen Sie den Schalter *Local Heap* im Dialogfenster *Windows Information*, und dann *OK*. Die Datenobjekte werden im *Log*-Fenster aufgelistet. Es ist, wie beim globalen Heap auch, zu beachten, daß nur eine bestimmte Anzahl von Zeilen in das *Log*-Fenster geschrieben werden kann. Vergleichen Sie dazu den Abschnitt »Auflistung des Inhalts des globalen Heaps« weiter oben.

Das folgende Beispiel zeigt eine typische Zeile aus der Auflistung über den Inhalt des lokalen Heap, gefolgt von einer Erklärung ihres Formats:

```
05CD0024 BUSY (10AF)
```

Feld	Beschreibung
05CD	Der Offset des Objekts im lokalen Datensegment
0024	Die Länge des Objekts in Bytes
BUSY	Die Disposition des Speicherobjekts, wie folgt:
	FREE Ein nicht belegter Block im Speicher
	BUSY Der Block ist zugewiesen
(10AF)	Ein lokales Speicher-Handle für das Objekt

Erstellung einer Liste der Module

Um eine Liste der Task- und DLL-Module zu erstellen, die von Windows geladen wurden, wählen Sie den Schalter *Module List* im Dialogfenster *Windows Information* und dann *OK*. Im *Log*-Fenster erscheint dann eine Liste der Module.

Im folgenden Beispiel sind drei Zeilen aus einer Modulliste aufgeführt, gefolgt von der Erklärung der letzten Zeile in der Liste:

```
0985 TASK TDW      C:\TD\TDW.EXE
0E2D DLL  WINDEBUG C:WIN3\WINDEBUG.DLL
0EFD TASK GENERIC  C:\TD\GENERIC.EXE
```

Feld	Beschreibung
0EFD	Ein Handle für ein Speichersegment, dargestellt als vierstelliger Hex-Wert.
TASK	Der Modultyp. Ein Modul kann entweder eine Task oder eine DLL sein.
GENERIC	Der Modulname
C:\TD\GENERIC.EXE	Der Pfad zu den ausführbaren Dateien des Moduls.

Debuggen von dynamischen Linkbibliotheken (DLLs)

Eine DLL ist eine Bibliothek von Routinen und Ressourcen, auf die Ihr Windows-Programm während der Laufzeit zugreifen kann (dynamisches Linken), anstatt schon bei der Compilierung (statisches Linken). Die Bindung zur Laufzeit ermöglicht es, daß mehrere Programme gleichzeitig auf die Routinen und Ressourcen der Bibliothek zugreifen können, wodurch Speicher gespart wird. Wenn ein Programm startet, das eine DLL verwendet, lädt Windows sie (wenn dies nicht schon vorher geschehen ist), so daß das Programm auf die jeweiligen Routinen in der DLL zugreifen kann.

Wenn Sie ein Programm in TDW laden, das auf DLLs zugreift, ermittelt TDW, welche dieser DLLs, wenn überhaupt, Symboltabellen haben (also mit eingeschalteter Debug-Option compiliert wurden). TDW protokolliert diese DLLs für Sie. Wenn TDW während der Ausführung Ihres Programms auf einen Aufruf einer Routine in einer dieser DLLs stößt, lädt es die Symboltabelle und die Quelldatei dieser DLL und setzt die Modulzeilenmarkierung an den Beginn der DLL-Routine, die vom Programm aufgerufen wurde. Die DLL ist dann genauso im Modul-Fenster verfügbar, wie es der Quelltext Ihrer Anwendung war.

Hinweis: TDW kann auch eine DLL laden, die keine Symboltabelle hat, allerdings nur in das CPU-Fenster.

Wenn die Routine der DLL abgearbeitet wurde, lädt TDW, wenn möglich, wieder die Symboltabelle und den Quelltext Ihres Programms und setzt die Zeilenmarkierung auf die nächste Anweisung nach dem Aufruf der DLL-Routine.

Hinweis: Wenn Sie den Ablauf Ihres Programms mittels F7 oder F8 schrittweise verfolgen, ist es für TDW eventuell nicht möglich, zur aufrufenden Routine in Ihrem Programm zurückzukehren, da die DLL über einen Windows-Funktionsaufruf zurückkehren könnte. In diesem Fall läuft Ihr Programm so weiter ab als ob Sie F9 gedrückt hätten. Dieses Verhalten ist öfters im Startup-Code einer DLL zu finden. Um eine Rückkehr zu Ihrem Programm zu erzwingen, setzen Sie, bevor Sie den Ablauf schrittweise verfolgen, in Ihrer Anwendung einen Breakpoint in der Zeile nach dem Aufruf an die DLL. Wenn Sie den Startup-Code einer DLL debuggen, setzen Sie den Breakpoint in der ersten Zeile des Programms.

Da das Debuggen von DLLs mit TDW weitgehend automatisch abläuft, müssen Sie nicht angeben, welche DLLs geladen werden sollen. Trotzdem können Sie folgende Arbeitsschritte durchführen:

- Hinzufügen einer DLL zu der DLL-Liste.

- Setzen von Breakpoints, Watches etc. in einer DLL.
- Angabe von DLLs, für die TDW keine Symboltabelle laden soll.
- Debuggen des Startup-Codes einer DLL.

Um diese Arbeitsschritte durchzuführen, müssen Sie das Dialogfenster *Load module source or DLL symbols* mit dem Befehl *View/Modules* aufrufen (wenn Sie F3 drücken, öffnet sich ebenfalls dieses Dialogfenster).

Das Dialogfensters *Load module source or DLL symbols*

Über dieses Dialogfenster stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Wechseln zu einer anderen Quelldatei Ihres Programms.
- Bearbeitung von DLL- und .EXE-Dateien (zum Beispiel Laden einer Symboltabelle oder Quelldatei).

Wechseln zu einer anderen Quelldatei

Wenn Sie ein Programm debuggen, das aus verschiedenen Quelldatei besteht, die zu einer .EXE-Datei gelinkt wurden, und Zugriff auf ein Modul des Programms brauchen, das sich gerade nicht im Modulfenster befindet, können Sie das Dialogfenster *Load module source or DLL symbols* aufrufen und ein Modul aus der Liste auf der linken Seite (der Liste der Quelldateien) wählen.

Alle Module, die in Ihr Programm einbezogen werden, sind hier aufgeführt.

Um eine Datei auszuwählen, heben Sie es hervor und drücken dann Enter, klicken zweimal mit der Maustaste auf dem Dateinamen oder wählen den Schalter *Load*. TDW zeigt daraufhin das Modulfenster mit der gewählten Quelldatei.

Arbeiten mit DLLs und Programmen

Wenn Sie ein Programm debuggen, das eine oder mehrere DLLs verwendet (wie die meisten Windows-Programme) und das Dialogfenster *Load module source or DLL symbols* aufrufen, wird in der Liste der Programme und DLLs (die rechte Liste) eine Aufstellung der DLL- und .EXE-Dateien angezeigt. Diese Liste umfaßt alle augenblicklich von Windows geladenen DLL- und .EXE-Dateien, und ebenfalls DLLs, die geladen werden, wenn Ihr Programm startet. Es enthält keine DLLs, die von Ihrem Programm mit dem Aufruf der Windows-Funktion *LoadLibrary* gestartet werden, außer eine der DLLs ist schon von Ihrem Programm oder von Windows geladen.

Die Elemente am Anfang der Liste (rechts mit einem Oval gekennzeichnet) sind die .EXE-Datei Ihres Programms und die DLLs mit Symboltabellen, die Ihr Programm verwendet. Wenn Sie keine Veränderungen vornehmen, lädt TDW automatisch die Symboltabelle und die Quelldatei für jede markierte DLL immer wenn Ihre Anwendung eine Routine dieser DLL aufruft. Zusätzlich lädt TDW automatisch die Symboltabelle und die Quelldatei jeder DLL, die Ihr Programm über einen Aufruf der Funktion *LoadLibrary* startet, selbst wenn die DLL eventuell zuerst nicht in der Liste erscheint (sie erscheint nachdem TDW sie geladen hat).

Die Schalter auf der rechten Seite der Liste bearbeiten die DLL oder das Programm, das Sie markiert haben. Im Texteingabefenster unter der Liste können Sie eine DLL in die Liste einfügen. Die Funktionen werden wie folgt verwendet:

Schalter	Beschreibung
Symbol load	Lädt die Symboltabelle und Quelldateien für die DLL oder das Programm, ungeachtet der Einstellungen von *Load Symbols*. Dieser Befehl setzt diese Einstellungen außer Kraft und ändert den Inhalt des Fensters *Module*, so daß Sie Breakpoints, Windows-Botschaften etc. für DLLs und Programme einstellen können.
Load symbols (No/Yes)	Wählen Sie, ob die DLL-Symboltabelle und die Quelldatei geladen werden sollen, wenn das Programm eine Routine der DLL aufruft. Sie können diese Option verwenden, um zu verhindern, daß TDW die Symboltabelle und die Quelldatei einer DLL lädt, die gar nicht untersucht werden muß. Die Voreinstellung ist *Yes*. Wenn Sie ein Programm erneut laden, ist *Load symbols* für alle DLLs und Module auf *Yes* eingestellt, auch für solche DLLs oder Module, die vorher auf *No* gesetzt wurden.
Debug startup (No/Yes)	Wählen Sie, ob der Startup-Code einer DLL untersucht werden soll. Die Voreinstellung ist *No*. Diese Schalter haben nur für DLLs eine Bedeutung. Um den Startup-Code von Programmen zu debuggen, müssen Sie TDW mit der Kommandozeilenoption -l starten.
DLL Name	Geben Sie den Namen einer DLL ein, die nicht in der Liste aufgeführt ist. Eine so hinzugefügte DLL kann dann mit den drei oben aufgeführten Befehlen bearbeitet werden. Hier kann falls notwendig ein voller Pfadname angegeben werden.
Add DLL	Damit fügen Sie im Eingabefeld eine DLL hinzu.

Eine DLL zur Liste DLLs & Programms hinzufügen

Bevor Sie die Debug-Optionen einstellen, den Startup-Code einer DLL untersuchen oder verhindern können, daß TDW die Symboltabelle und die Quelldatei einer DLL lädt, muß sich die DLL zuerst in der Liste der DLLs und Programme aufgeführt werden. Eine DLL, auf die von Ihrem Programm aus zugegriffen wird, ist eventuell nicht in der Liste aufgeführt, da nach dem Laden Ihres Programms TDW nur die DLLs kennt, die mit dem Startup-Code des Programm verbunden sind. Ihr Programm kann auch ausdrücklich eine DLL starten, indem es die Windows-Funktion *LoadLibrary* aufruft. TDW weiß erst dann über diese DLL Bescheid, wenn Ihr Programm *LoadLibrary* aufruft.

Hinweis: Es gibt zwei verschiedene Typen von Startup-Code: der Startup-Code Ihrer Anwendung und der Startup-Code der DLL. Manche DLLs werden innerhalb des Startup-Codes eines Programms gestartet. Wenn Ihr Programm eine DLL startet, wird zunächst der Startup-Code der DLL ausgeführt.

Darüber hinaus gibt es auch zwei Arten von Startup-Code einer DLL. Darauf werden wir weiter unten näher eingehen.

Wenn Sie eine DLL zur Liste *DLL & Programs* hinzufügen wollen, rufen Sie das Dialogfenster *Load Modules or DLLs* auf (mit F3 oder dem Menübefehl *View/Modules*), setzen den Cursor in das Eingabefeld *DLL Name*, geben den

Namen der DLL ein (falls nötig mit vollständiger Pfadangabe), und fügen ihn mit dem Schalter *Add DLL* in die Liste ein.

Einstellen der Debug-Optonen in einer DLL

Wenn Sie Breakpoints oder Watches oder andere Debug-Optionen für eine DLL einstellen wollen, rufen Sie das Dialogfenster *Load Modules or DLLs* (mit F3 oder über die Menübefehle *View/Modules*), markieren die DLL in der Liste, wählen dann *Symbol Load* und rufen so das Fenster *DLL in a Module* auf. Sobald Sie sich in diesem Fenster befinden, können Sie die DLL bearbeiten.

Wenn Sie die Untersuchung der DLL beendet haben, müssen Sie, bevor Sie wieder zum Debuggen Ihres Programms zurückkehren können, die Symboltabelle und die Quelldatei Ihres Programms laden. Rufen Sie das Dialogfenster *Load Modules or DLLs* auf, wählen Sie die .EXE-Datei Ihres Programms und drücken Sie den Schalter *Symbol Load*.

Laden der DLL-Symboltabellen

Als Voreinstellung lädt TDW die Symboltabelle und die Quelldatei jeder DLL, auf die Ihr Programm zugreift, jedoch nur, wenn die DLL eine TDW-kompatible Symboltabelle besitzt. Eine DLL hat eine zu TDW kompatible Symboltabelle, wenn es mit eingeschalteter Debug-Option von Turbo Pascal für Windows compiliert wurde.

Da es eine gewisse Zeit dauert, bis die Symboltabelle einer DLL und dann, nach der Abarbeitung der Routine in der DLL, wieder die Symboltabelle des Originalprogramms geladen wird, sollten Sie diese Ladeaktionen für DLLs ausschalten, die nicht untersucht werden müssen. Um zu verhindern, daß TDW eine Symboltabelle lädt, rufen Sie das Dialogfenster *Load Modules or DLLs* auf (mit F3 oder dem Menübefehl *View/Modules*), suchen die DLL in der Liste, markieren sie und stellen dann den Schalter *Load Symbols* auf *No*.

Debuggen von DLL Startup-Code

Gemäß Voreinstellung führt TDW den Startup-Code einer DLL aus und lädt deren Symboltabelle, wenn das Programm eine Routine in der DLL aufruft. TDW öffnet dann das Fenster *Module* oder *CPU* mit der aktuellen Zeilenmarkierung am Beginn der DLL-Routine, die vom Programm aufgerufen wurde.

TDW ermöglicht aber die Untersuchung des Startup-Codes, wenn Sie das Programm dazu anweisen. Sie können TDW dazu verwenden, zwei Arten von Startup-Code zu debuggen:

Der Initialisierungscode gleich nach *LibMain* (die Voreinstellung). Der in der DLL eingebundene Assembler-Code, der die DLL initialisiert und der auch jene Routinen umfaßt, die Umrechnungen entsprechend dem Speichermodell, das die DLL verwendet, vorzunehmen hat (wird ausgewählt, wenn Sie TDW mit der Kommandozeilenoption -l starten).

Nachdem Sie für eine oder mehrere DLLs in Ihrem Programm angegeben haben, daß Sie auch deren Startup-Code untersuchen wollen, lädt TDW die Symboltabelle für jede DLL entweder wenn der Startup-Code Ihres Programms die DLL lädt oder wenn Ihr Programm die Funktion *LoadLibrary* aufruft.

Wenn Sie sich erst nach dem Laden Ihres Programms für die Untersuchung des Startup-Codes von DLLs entscheiden, verhält sich TDW eventuell nicht wie erwartet, da einige oder alle der relevanten DLLs bereits geladen sind. Deswegen sollten Sie einen der beiden folgenden Wege beschreiten, wenn Sie den Startup-Code von DLLs debuggen wollen:

Wählen Sie die zu untersuchenden DLLs aus, bevor Sie das Programm laden.

Laden Sie Ihr Programm, wählen die DLLs aus und starten dann Ihr Programm noch einmal (über [Ctrl][F2] oder den Menübefehl *Run/Programm Reset*)

Unter Beachtung all dieser Regeln, können Sie mit den folgenden Arbeitsschritten angeben, daß Sie den Startup-Code einer oder mehrerer DLLs debuggen wollen:

1. Rufen Sie das Dialogfenster *Load Modules or DLLs* auf (mit F3 oder über die Menübefehle *View/Modules*).

2. Suchen Sie in der Liste die gewünschte DLL und markieren Sie sie.

3. Stellen Sie *Debug Startup* auf *Yes*.

4. Wiederholen Sie gegebenenfalls die Schritte 2 und 3 für weitere DLLs.

5. Wenn sich eine DLL nicht in der Liste befindet hat oder wenn in der Liste überhaupt keine DLLs aufgeführt sind (da das Programm noch nicht geladen ist), verwenden Sie das Eingabefeld *DLL Name* um den Namen der relevanten DLL einzugeben und dann mit dem Schalter *Add DLL* in die Liste einzufügen. Darauf können Sie den Schalter *Debug Startup* auf *Yes* setzen.

6. Wenn Sie alle DLLs angegeben haben, deren Startup-Code Sie untersuchen wollen, ist der nächste Schritt die Auswahl des Befehls *File/Load* um das Programm zu laden (wenn es noch nicht geladen ist) oder *Run/Program Reset* um Ihr Programm erneut zu laden (wenn es schon geladen war, bevor Sie die geschilderten Schritte ausgeführt haben).

7. Wenn Ihr Programm eine DLL startet, öffnet TDW entweder das Fenster *Module* und unterlegt die Zeile mit der *LibMain*-Routine der DLL (Voreinstellung) oder das CPU-Fenster und stellt den Beginn des Assemblercode-Listings für die Startup-Bibiliothek dar (da TDW mit der Option -l gestartet wurde).

8. Wenn Sie die Untersuchung des Startup-Codes einer DLL abgeschlossen haben, drücken Sie F9, um bis ans Ende des Startup-Codes zu gehen und zum Programm zurückzukehren. Wenn Sie weitere DLLs zur Untersuchung ihres Startup-Codes angemeldet haben, verfährt TDW auf gleiche Weise auch mit diesen DLLs, wenn Ihr Programm sie startet.

Hinweis: Stellen Sie sicher, daß Sie bis zum Ende des Startup-Codes einer DLL gehen, bevor Sie das aktuelle Programm erneut oder ein anderes Programm laden. Wenn Sie dies nicht tun, kann der nur teilweise ausgeführte Startup-Code zum Absturz von Windows führen, so daß Sie eventuell den Rechner erneut starten müssen.

Umwandlung von Speicher-Handles in Adressen

Windows verwendet Handles statt Adressen für Speicherobjekte, da Windows den Speicher selbst verwaltet und die physikalische Position eines Objekts im Speicher verändern kann. Wenn Sie die tatsächliche Adresse benötigen, die durch einen Speicher-Handle angegeben wird, können Sie die in TDW definierten Typumwandllungssymbole lh2fp (für ein lokales Handle) und gh2fp (für ein globales Handle) verwenden, um ein Handle zu dereferenzieren.

Sie können diese in TDW definierten Typumwandlungssymbole genauso verwenden wie die in Turbo Pascal für Zeiger anwendbaren Symbole. Den Typ des lokalen Speicher-Handles *HLocalMemory* können Sie zum Beispiel mit zwei Methoden umwandeln:

- Im Dialogfenster *Data/Inspect* kann der Ausdruck lh2fp(HLocalMemory) ausgewertet werden.
- Über den Befehl *Type Cast* im Fenster *Inspector* kann lh2fp als Typ eingegeben werden.

In beiden Fällen erhalten Sie als Ergebnis einen Zeiger, der auf das erste Zeichen des Speicherblocks zeigt, der über das Handle *HLocalMemory* referenziert wurde.

Sie können beide Techniken verwenden um kompliziertere Umwandlungen durchzuführen, zum Beispiel um eine mehrphasige Umwandlung durchzuführen. Dazu folgendes Beispiel:

```
Mystruct (lh2fp(HlocalMemory))^
```

12. Debuggen auf Assemblerebene

Dieses Kapitel richtet sich an Programmierer, die mit der Assemblerprogrammierung der 80x86-Prozessoren vertraut sind. Sie müssen die in diesem Kapitel beschriebenen Möglichkeiten nicht verwenden, wenn Sie Ihre Programme debuggen wollen, obwohl einige raffinierte Fehler mit den hier beschriebenen Methoden leichter gefunden werden können.

Wir erklären, wann es für Sie sinnvoll ist, beim Debuggen auf die Assembler-Ebene zurückzugreifen. Anschließend beschreiben wir das *CPU*-Fenster mit dem integrierten Disassembler und Assembler. Sie werden sehen, wie man reine Hex-Daten untersucht und verändert, wie der Stack mit den Funktionsaufrufen eingesetzt wird und wie CPU-Register und CPU-Flags untersucht und verändert werden.

Grenzen des Debuggens auf Quellcode-Ebene

Wenn Sie ein Programm debuggen, dann greifen Sie auf den Code und die Daten Ihres Programms meistens auf der Quellcode-Ebene zu. Sie verwenden die Symbolbezeichnungen genau so, wie Sie sie in Ihrem Quellcode angegeben haben. Sie wandern durch Ihr Programm, indem Sie Teile des Quellcodes abarbeiten.

Manchmal aber kann man ein Problem nur durch genaue Betrachtung der vom Compiler erzeugten Befehle, des Inhalts der CPU-Register und des Stack-Inhalts erfassen. Dazu müssen Sie aber mit den 80x86-Prozessoren vertraut sein und eine Vorstellung darüber haben, wie der Compiler Ihren Quellcode in Maschinensprachebefehle umwandelt. Da es sehr viele ausgezeichnete Bücher über die interne Funktionsweise der CPU gibt, werden wir hier nicht ins Detail gehen. Sie können sehr schnell lernen, wie der Compiler Ihren Quellcode in die Maschinensprachebefehle umwandelt, indem Sie sich die Befehle, die für jede einzelne Zeile im Quellcode erzeugt werden, genau anschauen.

In Pascal zum Beispiel können im Quellcode Zeilen geschrieben werden, die sehr viele Aktionen auf einmal ausführen. Da der Debugger jeweils eine ganze Zeile und nicht einen einzelnen Ausdruck abarbeitet, wenn man schrittweise durchs Programm geht, kann in der Auswertung nicht, wie vielleicht gewünscht,

jeder einzelne Schritt nachvollzogen werden. Indem jeder Maschinensprachebefehl einzeln abgearbeitet wird, kann man alle Zwischenresultate kontrollieren. Dazu muß man jedoch verstehen, wie der Compiler den Quelltext in den Maschinencode umgesetzt hat.

Das CPU-Fenster

Das *CPU*-Fenster gibt einen vollständigen Überblick über den Zustand der CPU. Sie können die Bits und Bytes, aus denen die Daten und der Code Ihres Programms bestehen, direkt untersuchen und verändern. Mit dem integrierten Assembler kann das Programm im Code-Ausschnitt temporär gepatcht werden. Die Befehle werden wie in einem Assemblerquelltext eingegeben. Sie können außerdem auf alle Bytes der damit zusammenhängenden Datenstrukturen zugreifen und diese in einer Reihe von Formaten darstellen und verändern.

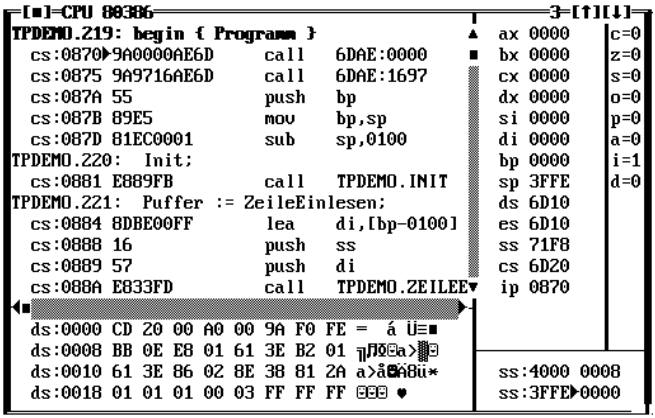


Abbildung 12.1
Das CPU-Fenster

Ein *CPU*-Fenster kann von der Menüleiste aus mit dem Befehl *View/CPU* geöffnet werden. Je nachdem, was Sie im aktuellen Fenster dargestellt haben, erscheint das *CPU*-Fenster mit der entsprechenden Stelle im Code-, Daten- oder Stack-Bereich. Dies ist eine bequeme Möglichkeit, an der jeweiligen Stelle im Code-, Daten- oder Stack-Bereich unter die Oberfläche zu gehen und alle Informationen detailliert anzuschauen.

Der folgenden Tabelle können Sie entnehmen, wo der Cursor stehen wird, wenn Sie den CPU-Befehl eingeben:

Aktuelles Fenster	CPU-Ausschnitt	Position
Stack-Fenster	Stack	Aktueller SS:SP
Module-Fenster	Code	Aktueller CS:IP
Variables-Fenster	Daten*	Adresse des Datenobjekts
Inspector-Fenster	Daten	Adresse des Datenobjekts
Breakpoint (kein globaler)	Code	Adresse des Breakpoints

* Codeausschnitt, wenn das Datenobjekt eine Routine ist

Die oberste Zeile des *CPU*-Fensters zeigt den Prozessor-Typ Ihres System an (8086, 80286, 80386 oder i486). Das *CPU*-Fenster besteht aus fünf Ausschnitten. Um von einem Ausschnitt zum nächsten zu gehen, drücken Sie Tab oder [Shift][Tab]. Mit der Maus brauchen Sie nur in den gewünschten Ausschnitt zu klicken. Der linke obere Ausschnitt (Code-Ausschnitt) zeigt den disassemblierten Programmcode, in dem zwischendurch die entsprechenden Zeilen des Quellcodes stehen. Der zweite Ausschnitt oben (Registerausschnitt) zeigt den Inhalt der CPU-Register. Der rechte Ausschnitt ist der Statusausschnitt, in dem der Zustand der acht CPU-Flags dargestellt wird. Der linke untere Ausschnitt (Datenausschnitt) enthält den reinen Hex-Dump eines ausgewählten Speicherbereichs. Der rechte untere Ausschnitt (Stack-Ausschnitt) zeigt den Inhalt des Stacks.

Im Code-Ausschnitt zeigt ein Pfeil die aktuelle Programmposition (CS:IP) an. Im Stack-Ausschnitt zeigt der Pfeil auf den aktuellen Stackzeiger (SS:SP).

Wenn sich der hervorgehobene Befehl im Code-Ausschnitt auf eine Speicherstelle bezieht, wird die Adresse und der aktuelle Inhalt der Speicherstelle in der obersten Zeile des CPU-Fensters angezeigt. Daraus kann man ersehen, auf welche Speicherstelle sich ein Befehlsoperand bezieht und welcher Wert in dieser Speicherstelle gelesen oder überschrieben wird.

Der Statusausschnitt zeigt die Einstellung der einzelnen CPU-Flags.

Wie alle anderen lokalen Menüs wird auch das lokale Menü des Code-Ausschnitts mit der Tastenkombination [Alt][F10] aktiviert. Wenn die Tastenkürzel mit den Ctrl-Tasten eingeschaltet sind, kann mit Hilfe der Taste Ctrl und dem ersten Buchstaben des Befehls direkt der gewünschte Befehl aufgerufen werden.

In den Code-, Daten- und Stack-Ausschnitten läßt sich die erste dargestellte Adresse des Ausschnitts mit den Tastenkombinationen [Ctrl][↓] und [Ctrl][↑] um ein Byte nach oben oder unten verschieben. Für eine geringfügige Anpassung der Darstellung ist das leichter als die Verwendung des Befehls *Goto*.

Der Code-Ausschnitt

In diesem Ausschnitt werden die disassemblierten Befehle an der von Ihnen vorgegebenen Adresse dargestellt.

Der linke Teil jeder disassemblierten Zeile enthält die Adresse des Befehls. Die Adresse wird entweder hexadezimal als Segment und Offset dargestellt, oder der Wert des Segments wird mit CS angegeben, wenn das Segment mit dem

aktuellen Code-Segment übereinstimmt. Wenn das Fenster breit genug ist (gezoomt oder vergrößert), werden auch noch die Bytes der einzelnen Maschinensprachebefehle angegeben. Der disassemblierte Befehl erscheint auf der rechten Seite.

Der Disassembler

Im Codeausschnitt werden Ihre Programmbefehle automatisch disassembliert und dargestellt. Wenn eine Adresse einem globalen oder statischen Symbol beziehungsweise einer Zeilennummer entspricht, enthält die Zeile vor der disassemblierten Anweisung die Quellcodezeilen, wenn als Darstellungsart beim Befehl *Mixed* die Option *Yes* angegeben wurde. Wenn die Quellcodezeile der Symboladresse entspricht, wird diese auch hinter dem Symbol dargestellt.

Globale Symbole werden einfach in Form der Symbolbezeichnung angegeben. Die statischen Symbole erscheinen als Modulnamen, gefolgt von einem Nummernzeichen (#) oder einem Punkt (.). Zeilennummern werden hinter dem entsprechenden Modulnamen angegeben. Dazwischen steht noch ein Nummernzeichen (#) oder ein Punkt (.). Die Zeilennummern werden dezimal angegeben.

Wenn ein konstanter Operand dargestellt wird, kann man den Datentyp aus der Anzahl der Ziffern ermitteln: ein Byte hat zwei Ziffern, ein Wort besteht aus vier Ziffern.

Turbo Debugger kann 8087/80287/80387-Koprozessoren automatisch entdecken und disassembliert diese Befehle entweder für den Koprozessor-Chip oder für den Emulator.

Der Befehl RETF deutet an, daß ein Rücksprung als FAR erfolgt. Der normale RET-Befehl bedeutet einen NEAR-Rücksprung.

Falls möglich, werden die Zieladressen von JMP- und CALL-Befehlen symbolisch dargestellt. Wenn CS:IP bei einem JMP- oder bedingten Sprungbefehl steht, deutet ein Pfeil nach oben und unten an, daß die Verzweigung im Befehl durchgeführt wird. Speicheradressen, die in MOV, ADD und anderen Befehlen verwendet werden, werden ebenfalls symbolisch dargestellt.

Das lokale Menü des Code-Ausschnitts

Wenn Sie mit dem Cursor nicht im Codeausschnitt landen, können Sie die Tasten Tab und [Shift][Tab] verwenden, um dorthin zu gelangen. Mit [Alt][F10] aktivieren Sie das lokale Menü.

Goto

Nachdem Sie den Befehl *Goto* (engl. gehe zu) ausgewählt haben, werden Sie nach der neuen Adresse gefragt. Sie können auch Adressen außerhalb Ihres Programmbereichs verwenden, um den Code im ROM-BIOS, in MS-DOS oder

in speicherresidenten Programmen zu untersuchen. In Kapitel 9 »Ausdrücke« erhalten Sie eine vollständige Beschreibung, wie Adressen eingegeben werden können.

Der Befehl *Previous* bringt den Code-Ausschnitt wieder dorthin zurück, wo er vor dem Aufruf eines *Goto*-Befehls gewesen ist.

Origin

Mit dem Befehl *Origin* (engl. Ursprung) kommt man zur aktuellen Programmposition, wie Sie im Registerpaar CS:IP steht. Dieser Befehl ist sehr praktisch, um wieder zum Ausgangspunkt zurückzukehren.

Der Befehl *Previous* bringt den Code-Ausschnitt wieder dorthin zurück, wo er vor dem Aufruf eines Origin-Befehls gewesen ist.

Follow

Der Cursor wechselt beim Befehl *Follow* (engl. [ver]folgen) zur Zieladresse des hervorgehobenen Befehls. Der Codeausschnitt steht an der Stelle im Programmcode, die im hervorgehobenen Befehl adressiert wird. Bei bedingten Sprungbefehlen wird davon ausgegangen, daß die Verzweigung ausgeführt wird.

Dieser Befehl kann im Zusammenhang mit allen CALL-Befehlen, JMP- Befehlen, den bedingten Verzweigungen (`JNZ`, `JO`, `LOOP`, `JCXZ` usw.) und den INT-Befehlen verwendet werden.

Mit dem Befehl *Previous* wird der Codeausschnitt wieder an die Position zurückgesetzt, die vor dem *Follow*-Befehl angezeigt wurde.

Caller

Der Codeausschnitt wechselt beim Aufruf von Caller (engl. Aufrufender) zu dem Befehl, der die aktuelle Routine aufgerufen oder den Interrupt ausgelöst hat.

Der Befehl funktioniert nicht immer. Wenn die Interrupt-Routinen oder das Unterprogramm selbst einige Werte auf den Stack gelegt haben, kann Turbo Debugger nicht herausfinden, woher der Aufruf kam.

Mit dem Befehl Previous wird der Codeausschnitt wieder an die Position zurückgesetzt, die vor dem Befehl Caller angezeigt wurde.

Previous

Mit dem Befehl Previous (engl. vorherige [Position]) wird der Code-Ausschnitt wieder an die Adresse zurückgebracht, die der Code-Ausschnitt vor dem letzten Aufruf einer ausdrücklichen Adreßänderung inne hatte. Wenn man den Code-Ausschnitt mit den Pfeiltasten oder [PgUp] und [PgDn] verändert, wird die Position nicht festgehalten. Mit Previous kommt man dann nicht zurück.

Wenn Sie den Befehl Previous auswählen, dann merkt sich Turbo Debugger die Position des Code-Ausschnitts wieder, so daß mit wiederholten Previous-Befehlen zwischen zwei Adressen umgeschaltet werden kann.

Search

Man kann mit *Search* (engl. suchen) einen Befehl oder eine Byte-Folge zum Suchen eingeben. Die Eingabe der Daten erfolgt wie beim Befehl *Assemble*.

Mit den gesuchten Befehlen sollten Sie vorsichtig sein. Sie sollten nur nach Befehlen suchen, die die gesuchten Bytes nicht verändern, wenn sie assembliert werden. Die Suche nach den folgenden Befehlen ist kein Problem:

```
push dx
pop  [di+4]
add  ax,100
```

Wenn Sie aber die folgenden Befehle suchen, kann es zu unvorhersehbaren Ergebnissen kommen:

```
je   123
call funker
loop $-100
```

Anstelle eines Befehls kann man auch eine Byte-Folge eingeben. In Kapitel 9 »Ausdrücke« erhalten Sie ausführliche Informationen über die Möglichkeiten der Eingabe von Byte-Folgen.

Mixed

Der Befehl *Mixed* (engl. gemischt[e Darstellung]) schaltet zwischen den drei möglichen Darstellungen von disassemblierten Befehlen und dem Quellcode um:

No Es wird überhaupt kein Quellcode dargestellt, sondern nur die disassemblierten Befehle.

Yes Die Zeilen des Quellcodes erscheinen vor dem ersten disassemblierten Befehl dieser Quellzeile. Der Ausschnitt wird mit dieser Darstellungsart aktiviert, wenn das aktuelle Modul in einer höheren Programmiersprache geschrieben ist.

Both Die Quellzeilen ersetzen die disassemblierten Zeilen mit dem entsprechenden Quellcode. Sonst erscheinen die disassemblierten Befehle. Dieser Modus ist anzuwenden, wenn man das Original im Quellcode sehen will und nicht den entsprechenden disassemblierten Befehl. Der Ausschnitt wird automatisch mit dieser Darstellungsart aufgerufen, wenn der Quelltext des aktuellen Moduls in Assembler geschrieben ist. In diesem Fall ergibt sich auch der größte Vorteil. In C oder Pascal bringt diese Darstellungsmethode wenig, da lediglich die Anweisungen mit Registervariablen direkt umgesetzt werden, alle anderen Anweisungen werden nämlich disassembliert.

New CS:IP

Die aktuelle Programmposition (CS:IP) wird mit *New CS:IP* auf die hervorgehobene Adresse versetzt. Wenn Sie Ihr Programm erneut laufen lassen, beginnt die Ausführung an dieser Adresse. Das ist sehr praktisch, um einen Code-Ausschnitt, der nicht ausgeführt werden soll, zu überspringen.

Achtung: Verwenden Sie diesen Befehl nur mit äußerster Vorsicht. Wenn Sie CS:IP an eine Stelle setzen, in der der Stack in einem anderen Zustand sein sollte als an der aktuellen Programmposition in CS:IP, dann kommt es mit fast hundertprozentiger Sicherheit zu einem Systemabsturz. Sie sollten diesen Befehl nicht dazu verwenden, um CS:IP an eine Adresse außerhalb der aktuellen Routine zu versetzen.

Assemble
Mit dem Aufruf von *Assemble* (engl. assemblieren) kann man einen Befehl assemblieren, der den Befehl an der momentan hervorgehobenen Stelle im Programm temporär ersetzt. Sie werden sofort nach dem Befehl gefragt, der assembliert werden soll. Weitere Einzelheiten finden Sie im Abschnitt »Der Assembler« im weiteren Verlauf dieses Kapitels.

Sie können diesen Befehl auch durch einfaches Eingeben des Assemblerbefehls aktivieren. Sie erhalten dann sofort ein Dialogfenster wie beim Befehl *Assemble*.

I/O
Im Menü I/O (engl. Abk. Ein-/Ausgabe) kann man einen Wert in den I/O-Adreßraum der CPU schreiben oder von dort auslesen. Damit kann auch der Inhalt von Ein-/Ausgaberegistern auf Zusatzplatinen gelesen und verändert werden. Nach der Eingabe des Befehls stehen vier Optionen bereit:

In byte (engl. Byte ein[lesen]) liest ein Byte von einer Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie ermitteln wollen. Mit Hilfe von *Read word* können Sie ein Wort aus einer Ein/Ausgabeadresse einlesen.

Out byte (engl. Byte aus[geben]) schreibt ein Byte in eine Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie festlegen wollen. Mit Hilfe von *Write word* können Sie ein Wort in eine Ein/Ausgabeadresse schreiben.

Read word (engl. Wort lesen) liest ein Wort von einer Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie ermitteln wollen. Mit Hilfe von *In byte* können Sie ein Byte aus einer Ein/Ausgabeadresse einlesen.

Write word (engl. Wort schreiben) schreibt ein Wort in eine Ein-/ Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie festlegen wollen. Mit Hilfe von *Out byte* können Sie ein Byte in eine Ein/Ausgabeadresse schreiben.

IN- und OUT-Befehle greifen auf den Ein-/Ausgabebereich zu, in dem sich unter anderem Geräte wie serielle Schnittstellen, Festplatten-Controller und Bildschirmadapter befinden.

Achtung: Mit diesen Befehlen sollten Sie vorsichtig sein. Einige Ein-/ Ausgabegeräte betrachten das Einlesen eines Ports als einen Befehlszugriff, bei dem zum

Beispiel Statusbits zurückgesetzt oder neue Daten eingelesen werden. Es kann sein, daß dadurch der normale Ablauf Ihres Programms nicht mehr gewährleistet ist.

Die Register- und Statusausschnitte

Der Registerausschnitt ist der obere Ausschnitt rechts vom Code-Ausschnitt. Er zeigt den Inhalt der CPU-Register.

Der rechte obere Ausschnitt ist der Statusausschnitt, in dem die Einstellung der acht CPU-Flags gezeigt wird. In der folgenden Tabelle sind die verschiedenen Flags mit ihrer Darstellung im Statusausschnitt aufgelistet:

Flag	Bezeichnung des Flags
c	Carry
z	Zero
s	Vorzeichen (Sign)
o	Überlauf (Overflow)
p	Parität
a	Hilfscarry (Auxiliary)
i	Interrupt
d	Direction

Das lokale Menü des Registerausschnitts

Mit [Alt][F10] erhalten Sie das lokale Menü des Registerausschnitts. Wenn die [Ctrl]-Tastenkürzel aktiviert sind, können Sie die einzelnen Menübefehle aber auch mit der Taste [Ctrl] und dem ersten Buchstaben des Befehls direkt aufrufen.

Increment
Der Befehl *Increment* (engl. erhöhen) erhöht den Wert des hervorgehobenen Registers um Eins. Dies stellt eine einfache Methode dar, um durch geringfügige Änderungen an Registerwerten falsche Indexgrenzen auszugleichen.

Decrement
Der Befehl *Decrement* (engl. erniedrigen) erniedrigt das gerade hervorgehobene Register um Eins.

Zero
Der Befehl *Zero* (engl. Null) setzt das hervorgehobene Register auf Null.

Change...
Mit dem Befehl *Change* (engl. verändern) kann der Wert des hervorgehobenen Registers verändert werden. Nach der Auswahl dieses Befehls werden Sie um den neuen Wert gebeten. Bei der Eingabe des Werts können Sie alle Möglichkeiten des Syntaxparsers voll ausschöpfen.

Sie können diesen Befehl auch aktivieren, indem Sie einfach mit der Eingabe des neuen Registerwerts anfangen. Wenn Sie das machen, erscheint sofort ein Dialogfenster mit einem Eingabefeld wie bei der Eingabe des Befehls *Change*.

Registers 32-bit
Der Befehl *Registers 32-bit* schaltet die Darstellung zwischen 16- und 32-Bit-Registerwerten um. Normalerweise erscheinen 16-Bit-Register, falls Sie diesen Befehl nicht verwenden, um die Darstellung auf 32-Bit umzustellen. Sie brauchen die 32-Bit-Register nur, wenn Sie ein Programm debuggen, das tatsächlich die 32-Bit-Adressierung des 80386-Chips ausnützt. Wenn Sie ein gewöhnliches Programm mit der normalen 16-Bit-Adressierung debuggen, können Sie auf die übersichtlichere 16-Bit-Darstellung der Register umschalten.

Das lokale Menü des Statusausschnitts

Mit [Alt][F10] erhalten Sie das lokale Menü des Statusausschnitts. Sie können die einzelnen Menübefehle aber auch mit der Taste Ctrl und dem ersten Buchstaben des Befehls aufrufen, wenn die Ctrl-Tastenkürzel aktiviert sind.

Toggle
Setzt den Wert des Flags auf 0, wenn er vorher 1 war, und auf 1, wenn er vorher 0 war. Der Wert 0 entspricht einem »gelöschten« Bit und 1 einem »gesetzten« Bit. Sie können auch einfach ↵ eingeben, um den Wert des hervorgehobenen Flags umzuschalten.

Der Datenausschnitt

Dieser Ausschnitt bietet eine ungefilterte Darstellung eines ausgewählten Speicherbereichs. Im linken Bereich wird die Adresse der in dieser Zeile dargestellten Daten angegeben. Die Adresse wird entweder in Form eines hexadezimal angegebenen Segments und Offsets dargestellt, oder der Segmentwert wird durch die Bezeichnung des DS-Registers ersetzt, wenn der Wert des Segments mit dem aktuellen DS-Segment übereinstimmt.

Dahinter werden die Daten von einem oder mehreren Datenobjekten dargestellt. Das Format dieses Bereichs hängt von der Darstellungsart ab, die mit dem Menübefehl *Display as* ausgewählt wurde. Wenn Sie eines der Fließkommaformate (*Comp*, *Float*, *Real*, *Double*, *Extended*) eingestellt haben, wird nur eine

einzige Fließkommazahl in einer Zeile dargestellt. Beim *Byte*-Format werden acht Bytes in einer Zeile dargestellt, beim Format *Word* werden vier Worte pro Zeile dargestellt und beim *Long*-Format zwei Doppelworte.

Im rechten Teil der Darstellung werden die Zeichen gezeigt, die den Bytes entsprechen. Turbo Debugger gibt zu allen Byte-Werten das entsprechende Bildschirmzeichen aus. Sie brauchen deshalb nicht überrascht zu sein, wenn Sie seltsame Zeichen auf der rechten Seite des Speicher-Dumps sehen. Es sind dies lediglich die entsprechenden Bildschirmzeichen für die nebenstehenden Hexadezimalwerte.

Die Anzahl der Bytes, die in einer Zeile dargestellt werden, hängt vom Format ab, das mit dem Befehl *Display as* eingestellt wurde.

Achtung: Wenn Sie den Datenausschnitt zur Untersuchung des Bildschirmspeichers, der ROM-BIOS-Daten oder der Interrupt-Vektoren verwenden, dann sehen Sie die Werte des Programms, das untersucht wird, und nicht die tatsächlich vorhandenen Werte, wenn Turbo Debugger läuft. Dies sind nicht immer dieselben Werte, die zu dem Zeitpunkt, da Sie sie untersuchen, wirklich im Speicher stehen. Turbo Debugger merkt sofort, wenn Sie auf Speicherbereiche zugreifen, die er auch verwendet, und liefert Ihnen die richtigen Werte aus den Bereichen, in denen die Werte Ihres Programms gespeichert sind.

Das lokale Menü des Datenausschnitts

Wenn Sie mit dem Cursor im Datenausschnitt sind, können Sie mit [Alt][F10] das lokale Menü aufrufen. Sie können die einzelnen Menübefehle aber auch mit der Taste [Ctrl] und dem ersten Buchstaben des Befehls direkt aufrufen, wenn die [Ctrl]-Tastenkürzel aktiviert sind.

Goto

Mit dem Befehl *Goto* (engl. gehe zu) kommen Sie an eine bestimmte Adresse in Ihren Daten. Geben Sie die neue Adresse ein, die Sie sehen wollen. Sie können auch Adressen eingeben, die außerhalb Ihres Programms liegen und zum Beispiel von DOS, von speicherresidenten Programmen oder vom BIOS verwendet werden. In Kapitel 9 »Ausdrücke« wird ausführlich erläutert, welche Möglichkeiten für die Eingabe von Adressen zur Verfügung stehen.

Search

Der Befehl *Search* (engl. suchen) sucht einen String ab der aktuellen Speicheradresse. Geben Sie anschließend die gesuchte Byte-Folge ein. Wenn man zum Ende eines Segments kommt, wird die Suche nicht am Anfang des Segments fortgesetzt. In Kapitel 9 »Ausdrücke« finden Sie eine vollständige Erklärung der Syntax für Byte-Folgen.

Next

Mit Hilfe des Befehls *Next* (engl. Nächster) wird die nächste Position einer Byte-Folge gesucht, die Sie in einem *Search*-Befehl zuvor schon gesucht haben.

Change
Der Befehl *Change* ermöglicht die Veränderung von Bytes an der aktuellen Cursorposition. Wenn Sie in der ASCII-Darstellung sind oder als Format *Byte* ausgewählt wurde, dann werden Sie nach einer Byte-Folge gefragt. Ansonsten müssen Sie einen Wert für den aktuell dargestellten Datentyp eingeben. In Kapitel 9 »Ausdrücke« werden die Byte-Folgen ausführlich beschrieben.
Sie können diesen Befehl auch einfach dadurch aktivieren, daß Sie anfangen, den neuen Wert einzugeben. Daraufhin erhalten Sie ein Eingabefeld in einem Dialogfenster wie beim Befehl *Change*.

Follow
Durch die Auswahl des Befehls *Follow* (engl. [ver]folgen) erhalten Sie ein Menü, in dem Sie NEAR- oder FAR-Zeigerketten zurückverfolgen können.

Near code
interpretiert das Wort unter dem Cursor im Datenausschnitt als einen Offset im aktuellen Code-Segment, das im CS-Register steht. Der Code-Ausschnitt wird zum aktuellen Ausschnitt und zeigt die ausgewählte Adresse an.

Far code
interpretiert das Doppelwort unter dem Cursor im Datenausschnitt als eine FAR-Adresse (Segment und Offset). Der Code-Ausschnitt wird zum aktuellen Ausschnitt und zeigt die ausgewählte Adresse an.

Offset to data
erlaubt die Verfolgung von Zeigerketten, die aus einem Wort (NEAR, nur Offset) bestehen. Der Datenausschnitt wird auf den Offset positioniert, der im Wort in der Speicherstelle des Cursors steht.

Segment:offset to data
erlaubt die Verfolgung von Zeigerketten, die aus einem Doppelwort (FAR, Segment und Offset) bestehen. Der Datenausschnitt wird auf den Offset positioniert, der in den beiden Worten in der Speicherstelle des Cursors steht.

Base segment:0 to data
wertet das Wort unter dem Cursor als eine Segmentadresse aus und positioniert den Datenausschnitt an den Anfang dieses Segments.

Previous
Der Befehl *Previous* (engl. vorherige [Position]) positioniert den Datenausschnitt an die Stelle, die der Datenausschnitt vor der letzten ausdrücklichen Veränderung innehatte. Bei Veränderungen mit den Pfeiltasten oder PgUp und PgDn wird die letzte Position nicht festgehalten.
Turbo Debugger merkt sich die letzte Position auf einem Stack mit den letzten fünf Adressen, so daß man mehrere Aufrufe der Befehle des Menüs *Follow* oder von *Goto* zurückverfolgen kann.

Display as

Mit Hilfe des Befehls *Display as* (engl. Darstellung als) kann die Erscheinungsweise der Daten im Datenausschnitt festgelegt werden. Dabei kann man alle Formate verwenden, die von Pascal oder Assembler unterstützt werden. Sie können eine der nachfolgend beschriebenen Einstellungen auswählen.

Byte stellt den Datenausschnitt auf die Darstellung hexadezimaler Bytes um. Dies entspricht dem Datentyp byte in Pascal.

Word sorgt dafür, daß die Daten im Datenausschnitt in Form von hexadezimalen Worten ausgegeben werden. Dies entspricht dem Datentyp word in Pascal.

Long stellt den Datenausschnitt auf die Darstellung von hexadezimalen Doppelworten um. Dies entspricht dem Datentyp longint in Pascal.

Comp stellt den Datenausschnitt auf die Ausgabe von 8-Byte-Integerzahlen um. Der Wert wird dezimal angezeigt. Dies entspricht dem Datentyp comp (IEEE) in Pascal.

Float stellt den Datenausschnitt auf die Darstellung von Fließkommazahlen einfacher Genauigkeit ein. Die Werte werden in Exponentialdarstellung ausgegeben. Dies entspricht dem Datentyp single (IEEE) in Pascal.

Real stellt den Datenausschnitt auf die Darstellung von Pascal-Fließkommazahlen (sechs Bytes) um. Die Werte werden in Exponentialdarstellung ausgegeben. Dies entspricht dem Datentyp real in Pascal.

Double stellt den Datenausschnitt auf die Darstellung von Fließkommazahlen doppelter Genauigkeit (acht Bytes) um. Die Werte erscheinen in Exponentialdarstellung. Dies entspricht dem Datentyp double (IEEE) in Pascal.

Extended stellt die Ausgabe des Datenausschnitts auf Fließkommazahlen (10 Bytes) erweiterter Genauigkeit um. Die Werte erscheinen in Exponentialdarstellung. Dies entspricht dem Datentyp extended (IEEE) in Pascal.

Block

In Menü *Block* lassen sich ganze Speicherblöcke manipulieren. Sie können Speicherbereiche kopieren, löschen und initialisieren, sowie von Dateien lesen und in Dateien schreiben. Mit dem Befehl *Block* erhalten Sie die nachstehende Optionen.

Clear setzt einen zusammenhängenden Speicherbereich auf Null (0). Sie werden nach der Adresse und der Anzahl der Bytes gefragt, die gelöscht werden sollen.

Move kopiert einen Speicherblock von einer Adresse an eine andere. Sie werden nach der Quelladresse, der Zieladresse und der Anzahl der Bytes gefragt.

Set initialisiert einen zusammenhängenden Speicherbereich mit einem bestimmten Wert. Sie werden nach der Adresse und der Anzahl der Bytes gefragt, sowie nach dem Wert, auf den sie gesetzt werden sollen.

Read liest den Inhalt von Dateien ganz oder teilweise in den Speicher ein. Sie müssen zunächst den Namen der Datei, anschließend die Speicheradresse und dann die Anzahl der Bytes angeben.

Write schreibt einen Speicherbereich in eine Datei. Sie müssen zunächst den Namen der Datei, anschließend die Speicheradresse und dann die Anzahl der Bytes angeben.

Der Stack-Ausschnitt

In der rechten unteren Ecke des *CPU*-Fensters befindet sich der Stackausschnitt, in dem der Inhalt des Stacks angezeigt wird.

Das lokale Menü des Stack-Ausschnitts

Mit [Alt][F10] erhalten Sie das lokale Menü des Stack-Ausschnitts. Wenn die [Ctrl]-Tastenkürzel aktiviert sind, können Sie die einzelnen Menübefehle mit der Taste [Ctrl] und dem ersten Buchstaben des Befehls auch direkt aufrufen.

Goto
Mit dem Befehl *Goto* (engl. gehe zu) können Sie an eine bestimmte Stelle auf dem Stack springen. Geben Sie anschließend die neue Stackadresse ein. Wenn Sie wollen, können Sie auch Adressen außerhalb Ihres Programmstacks angeben, obwohl Sie für diese Zwecke auch den Datenausschnitt zur Verfügung haben, in dem Sie beliebige Speicherbereiche darstellen können. In Kapitel 9 »Ausdrücke« finden Sie eine Beschreibung, auf welche Art und Weise die Adressen eingegeben werden können.
Der Befehl *Previous* bringt den Stackausschnitt wieder an die Position zurück, an der der Stack vor dem Aufruf des letzten *Goto*-Befehls war.

Origin
Durch die Eingabe des Befehls *Origin* (engl. Ursprung) wird der Cursor wieder an die von SS:SP adressierte Speicherstelle zurückversetzt. Dieser Befehl ist sehr praktisch, wenn man wieder zu seinem Ausgangspunkt zurückkehren will.

Follow

Mit Hilfe des Befehls *Follow* (engl. [ver]folgen) werden Sie an die Stelle im Stack positioniert, die im gerade hervorgehobenen Wort adressiert wird. Diese Option eignet sich vor allem zum Zurückverfolgen zur aufrufenden Funktion über die Parameter am Stack.

Der Befehl *Previous* bringt den Stackausschnitt wieder an die Position, an der der Stack vor dem Aufruf des letzten *Origin*-Befehls war.

Previous

Der Befehl *Previous* (engl. vorherige [Position]) positioniert den Stackausschnitt an die Stelle, die der Stackausschnitt vor der letzten ausdrücklichen Veränderung des Stacks innehatte. Bei Veränderungen mit den Pfeiltasten oder PgUp und PgDn wird die letzte Position nicht festgehalten.

Wenn der Befehl *Previous* mehrmals hintereinander eingegeben wird, springt der Stack-Ausschnitt zwischen zwei Adressen hin und her.

Change...

Mit dem Befehl *Change* (engl. verändern) kann man in der hervorgehobenen Speicherstelle auf dem Stack einen neuen Wert eingeben.

Sie können diesen Befehl auch aktivieren, indem Sie einfach mit der Eingabe des neuen Registerwerts anfangen. Wenn Sie das machen, erscheint sofort ein Dialogfenster, als ob Sie den Befehl *Change* eingegeben hätten.

Der Assembler

Mit Turbo Debugger kann man Befehle für die Prozessoren 8086, 80186, 80286, 80386 und i486 sowie die Koprozessoren 8087, 80287 und 80387 assemblieren.

Wenn Sie den in Turbo Debugger integrierten Assembler verwenden, um Ihr Programm zu verändern, werden diese Änderungen nicht dauerhaft vorgenommen. Wenn Sie Ihr Programm mit dem Befehl *Run/Program reset* erneut laden oder ein anderes Programm mit dem Befehl *File/Open* laden, sind alle von Ihnen durchgeführten Veränderungen wieder verloren.

Üblicherweise verwendet man den Assembler, um eine Idee für eine bestimmte Problemlösung in einem Programm schnell auszuprobieren. Sobald Sie nachgeprüft haben, daß die Veränderung funktioniert, müssen Sie den Debugger verlassen, in Ihrem Quellcode alles nachtragen und anschließend Ihr Programm neu übersetzen und linken.

Der folgende Abschnitt beschreibt die Unterschiede zwischen dem integrierten Assembler und der Syntax, die von Turbo Pascals integriertem Assembler akzeptiert wird.

Vorgaben für die Größe von Adreßoperanden

Bei den Unterprogrammaufrufen (CALL), Sprungbefehlen (JMP) und den bedingten Verzweigungen (JNE, JL usw.) erzeugt der Assembler automatisch immer den kürzesten Befehl für die gegebene Entfernung. Mit Hilfe der Operandenvorgaben NEAR und FAR vor der Zieladresse läßt sich der Befehl mit einer bestimmten Größe assemblieren. Zum Beispiel:

```
CALL    FAR XYZ
JMP     NEAR A1
```

Konstanten und Speicheroperanden

Wenn Sie ein Symbol aus Ihrem Programm als Befehlsoperanden verwenden, müssen Sie dem integrierten Assembler mitteilen, ob Sie den Inhalt des Symbols oder die Adresse des Symbols wollen.

Wenn Sie ein Symbol, das in Ihrem Programm definiert ist, als Befehlsoperanden verwenden, müssen Sie dem integriertem Assembler mitteilen, ob Sie den Wert des Symbols oder dessen Adresse meinen. Geben Sie nur den Namen des Symbols an, wird es wie eine Adresse behandelt, so als ob der OFFSET-Operator verwendet würde. Setzen Sie dagegen das Symbol in eckige Klammern ([ ]), wird es zu einer Speicherreferenz. Wenn Ihr Programm zum Beispiel die Datendefinition

```
a       DW 4
```

enthält und Sie einen Befehl assemblieren oder einen Ausdruck auswerten wollen, der sich auf den Wert dieser Variablen bezieht, so benutzen Sie den Namen der Variablen allein oder in Klammern:

```
mov     ax,[a]
mov     dx,a
```

Um auf die Adresse der Variablen zuzugreifen, benutzt man den Operator OFFSET:

```
mov     ax,OFFSET a
```

Vorgaben für den Datentyp eines Operanden

Bei einigen Befehlen muß man die Operandengröße mit Hilfe eines der beiden folgenden Ausdrücke vor dem Operanden angeben:

```
BYTE PTR
WORD PTR
```

Zwei Beispiele für Befehle, in denen diese Vorgaben enthalten sind:

```
add     BYTE PTR [si],10
mov     WORD PTR [bp+10],99
```

Außer diesen beiden Datentypvorgaben kann man noch die folgenden Vorgaben verwenden, wenn man Befehle für den 80x87-Koprozessor assemblieren will:

```
DWORD PTR
QWORD PTR
TBYTE PTR
```

Zwei Beispiele für diese Vorgaben:

```
fild    QWORD PTR [bx]
fstp    TBYTE PTR [bp+4]
```

Stringbefehle

Wenn Sie einen Stringbefehl assemblieren, dann müssen Sie die Größe (Byte oder Word) in der Kurzform des Befehls angeben. Der Assembler akzeptiert keine Stringbefehle, in denen bei der Kurzform nur die Operanden mit ihren Datentypen stehen, bei denen aber im Befehl keine Angabe zur Datengröße enthalten ist. Sie müssen zum Beispiel STOSW anstelle von STOS WORD PTR[DI] hinschreiben.

Das Fenster Dump

Das *Dump*-Fenster zeigt Ihnen den reinen Auszug eines beliebigen Speicherbereichs. Es funktioniert genauso wie der Datenausschnitt im *CPU*-Fenster.

Im vorangegangenen Abschnitt »Das lokale Menü des Datenausschnitts« dieses Kapitels finden Sie eine Beschreibung des Inhalts und der lokalen Menüs für dieses Fenster.

Normalerweise verwendet man dieses Fenster, wenn man ein Assemblerprogramm auf Quellcode-Ebene untersucht und einen direkten Blick auf manche Datenbereiche werfen will. Mit dem Befehl *View/Dump* können Sie ein *Dump*-Fenster anlegen.

Sie können dieses Fenster auch in einem *Inspector*-Fenster verwenden, wenn Sie einen Blick auf die reinen Bytes Ihrer untersuchten Datenobjekte werfen

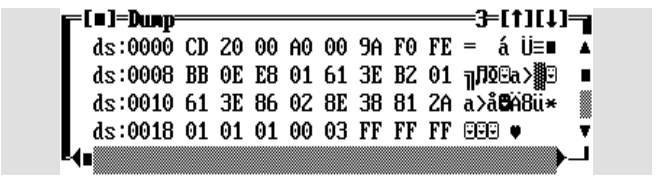
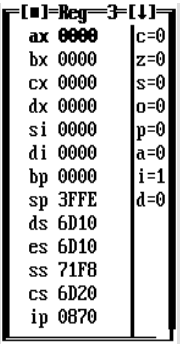


Abbildung 12.2 Das Dump-Fenster



Abbild 12.3 Das Fenster Registers

wollen. Verwenden Sie den Befehl *View/Dump*, um ein *Dump*-Fenster zu erhalten, das die im *Inspector*-Fenster stehenden Daten zeigt.

Das Fenster Registers

Das Fenster *Registers* zeigt den Inhalt der CPU-Register und der Flags. Es ist eine Kombination der Register- und Statusausschnitte des *CPU*-Fensters.

In den vorangegangenen Abschnitten »Das lokale Menü des Registerausschnitts« und »Das lokale Menü des Statusausschnitts« dieses Kapitels finden Sie eine Beschreibung des Inhalts und der lokalen Menüs in diesem Fenster.

Verwenden Sie dieses Fenster, wenn Sie ein Assemblerprogramm auf Quellcode-Ebene debuggen und die Registerwerte sehen wollen.

13. 80x87-Koprozessor und Gleitkomma-Emulator

Wenn Sie in Ihrem Programm Gleitkommazahlen verwenden, dann können Sie mit Turbo Debugger den Status des 80x87-Arithmetikprozessors bzw. des Emulators untersuchen und verändern. Dieses Kapitel richtet sich an Programmierer, die mit den Operationen des 80x87-Koprozessors bereits vertraut sind. Man muß die Möglichkeiten, die in diesem Kapitel zum Debuggen von Programmen mit Gleitkommazahlen beschrieben werden, nicht unbedingt ausnutzen, obwohl man einige vertrackte Fehler damit leichter finden kann.

In diesem Kapitel stellen wir die Unterschiede zwischen dem 80x87-Chip und dem Software-Emulator heraus. Wir beschreiben das Fenster *Numeric Processor* und zeigen Ihnen, wie man die Gleitkommaregister, sowie die Status- und Kontroll-Flags untersuchen und verändern kann.

Der 80x87-Chip im Vergleich zum Emulator

Turbo Debugger stellt automatisch fest, ob Ihr Programm den Arithmetikprozessor oder den Emulator verwendet, und paßt seine Arbeitsweise dementsprechend an.

Beachten Sie bitte, daß die meisten Programme entweder den Emulator oder den Koprozessor verwenden, aber nicht beide im selben Programm. Wenn Sie ein spezielles Assemblerprogramm geschrieben haben, das beide verwendet, dann ist Turbo Debugger nicht in der Lage, den Status des Koprozessors darzustellen. Es kann dann lediglich der Zustand des Emulators wiedergegeben werden.

Das Fenster Numeric Processor

Von der Menüleiste aus erzeugen Sie mit dem Befehl View/Numeric processor das Fenster Numeric Processor. Die oberste Zeile des Fensters enthält den aktuellen Programmzeiger (IPTR), den Zeiger auf Datenbereiche (OPTR) und den Befehl

als Hexadezimalcode (*OPCODE*). Der Datenzeiger und der Programmzeiger werden als physikalische 20-Bit-Adressen angegeben. Sie können diese Adressen in Segment:Offset-Adressen umwandeln, wenn Sie die ersten vier Ziffern als Segment auffassen und die letzte Ziffer als Offset verwenden.

Wenn in der obersten Zeile zum Beispiel IPTR=5A669 steht, kann dieser Wert in die Adresse 5A66:9 zerlegt werden, falls Sie den aktuellen Befehl und die entsprechenden Daten im *CPU*-Fenster untersuchen wollen. Das Fenster *Numeric Processor* gliedert sich in drei Abschnitte: Der linke Ausschnitt (Registerausschnitt) zeigt den Inhalt der Gleitkommaregister, der mittlere Ausschnitt (Kontrollausschnitt) enthält die Kontroll-Flags und der rechte Ausschnitt (Statusausschnitt) zeigt die Status-Flags.

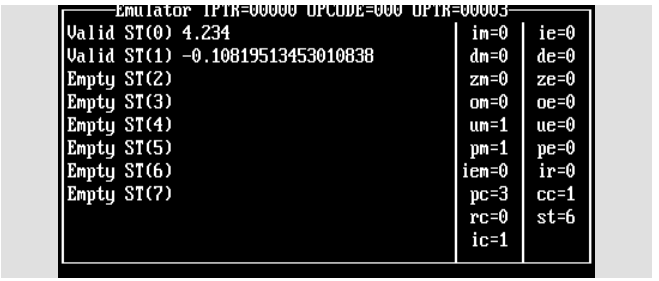


Abbildung 13.1 Das Fenster Numeric Processor

Die oberste Zeile enthält außerdem den letzten ausgeführten Gleitkommabefehl. In IPTR steht die physikalische 20-Bit-Adresse der letzten Gleitkommaoperation. In OPCODE wird angegeben, welcher Befehl ausgeführt wurde. Hinter OPTR finden wir die Speicherstelle, die mit diesem Befehl adressiert wurde, falls ein Speicheroperand im Befehl vorkam.

Der Registerausschnitt

Die 80-Bit-Gleitkommaregister

Im Registerausschnitt sind die einzelnen Gleitkommaregister (ST(0) bis ST(7) zusammen mit ihrem Status dargestellt (*valid* [zulässig] / *zero* [Null] / *special* [Sonderfall] / *empty* [leer]). Der Inhalt wird als eine 80-Bit-Gleitkommazahl angegeben.

Wenn Sie das Fenster *Numeric Processor* auf den gesamten Bildschirm ausgedehnt haben (durch Eingabe von F5) oder mit dem Befehl *Window/Move/Resize* verbreitert haben, dann sehen Sie die Gleitkommaregister auch in der reinen Hexadezimaldarstellung.

Das lokale Menü des Registerausschnitts

Um das lokale Menü des Registerausschnitts zu erhalten, drücken Sie die Tastenkombination [Alt][F10] oder rufen den gewünschten Befehl direkt mit der Ctrl-Taste zusammen mit dem ersten Buchstaben des Menüpunkts auf.

Zero löscht das gerade hervorgehobene Register auf den Wert 0.

Empty setzt das hervorgehobene Register auf den Status *empty*. Dieser spezielle Status zeigt an, daß das Register keine gültigen Daten mehr enthält.

Change Mit diesem Befehl kann das hervorgehobene Register mit einem neuen Wert geladen werden. Nach dem Aufruf des Befehls werden Sie nach dem entsprechenden Zahlenwert gefragt. Sie können den Wert ganzzahlig oder als Gleitkommazahl eingeben, wobei der komplette Ausdrucksparser der eingestellten Programmiersprache zur Verfügung steht. Der eingegebene Wert wird automatisch in das 80-Bit-Gleitkommaformat des Koprozessors umgewandelt.

Der Befehl kann auch ganz einfach dadurch aktiviert werden, daß man einen neuen Wert für das Gleitkommaregister eingibt. Wenn Sie das machen, erscheint nach der ersten Eingabe sofort ein Dialogfenster, so als ob Sie den Befehl *Change* eingegeben hätten.

Der Kontroll-ausschnitt

Die Kontroll-Flags

Die folgende Tabelle enthält die verschiedenen Kontroll-Flags und ihre Darstellung im Kontrollausschnitt:

Bezeichnung im Ausschnitt	Beschreibung des Flags
`im`	Bitmaske für unzulässige Operationen
`dm`	Bitmaske für denormalisierte Operanden
`zm`	Bitmaske für Division durch Null
`om`	Bitmaske für Überlauf
`um`	Bitmaske für Unterlauf
`pm`	Bitmaske für Genauigkeitsverluste
`iem`	Bitmaske für Interrupts (nur 8087)
`pc`	Genauigkeitskontrolle
`rc`	Rundungskontrolle
`ic`	Unendlichkeitsdarstellung

Das lokale Menü des Kontrollausschnitts

Mit Tab kommen Sie zum Kontrollausschnitt. Drücken Sie dann [Alt][F10], um das lokale Menü zu aktivieren. (Sie können auch die Ctrl-Taste mit dem ersten Buchstaben des Befehls verwenden, um den Befehl direkt aufzurufen.)

Toggle wandert durch die Werte, die man im gerade hervorgehobenen Kontroll-Flag einstellen kann. Die meisten Flags können nur gesetzt oder gelöscht werden (0 oder 1), so daß der Wert des Flags lediglich umgeschaltet wird. Einige andere Flags können mehr als zwei Werte annehmen. Bei diesen Flags wird der Wert des Flags solange inkrementiert, bis der Maximalwert erreicht wird. Anschließend wird wieder bei 0 angefangen.

Die Kontroll-Flags können auch mit der Taste ⏎ umgeschaltet werden.

Der Status-ausschnitt

Die Status-Flags

Die folgende Tabelle enthält die verschiedenen Status-Flags und ihre Darstellung im Statusauschnitt:

Bezeichnung im Ausschnitt	Beschreibung des Flags
ie	Unzulässige Operation
de	Denormalisierter Operand
ze	Division durch Null
oe	Überlauf
ue	Unterlauf
po	Genauigkeitsverlust
ir	Interrupt-Anfrage
cc	Bedingungscode
st	Zeiger auf Stack-Spitze

Das lokale Menü des Statusausschnitts

Mit Tab kommen Sie zum Statusausschnitt. Sie drücken dann [Alt][F10], um das lokale Menü zu aktivieren. (Sie können auch die Ctrl-Taste zusammen mit dem ersten Buchstaben des gewüschten Befehls verwenden, um den Befehl direkt aufzurufen.)

Toggle wandert durch die Werte, die man im gerade hervorgehobenen Status-Flag einstellen kann. Die meisten Flags können nur gesetzt oder gelöscht werden (0 oder 1), so daß der Wert des Flags lediglich umgeschaltet wird. Einige andere Flags können mehr als zwei Werte annehmen. Bei diesen Flags wird der Wert des Flags solange inkrementiert, bis der Maximalwert erreicht ist. Anschließend wird wieder bei 0 angefangen.

Die Status-Flags können auch mit ⏎ umgeschaltet werden.

14. Die Befehlsübersicht

In den vorangegangenen Kapiteln wurden alle Befehle beschrieben. Hier folgt nun eine kurze Zusammenfassung dieser Befehle. In diesem Kapitel beschreiben wir

- die Befehle, die mit einer einzigen Funktionstaste oder einer anderen Tastenkombination ausgelöst werden,
- die Befehle in der Menüleiste und den lokalen Menüs der einzelnen Fenster,
- alle Tastenkombinationen, die in den beiden Arten von Fensterausschnitten vorkommen (Text und Listen),
- die Befehle zum Anpassen von Fenstern.

Hotkeys

Ein Hotkey ist eine Taste, mit der eine Aktion unabhängig von der Position in Turbo-Debugger-Umgebung ausgeführt wird. In Tabelle 14.1 sind alle Hotkeys aufgeführt.

Tabelle 14.1 Die Funktionstasten und die Hotkeys

Taste	Menübefehl	Funktion
F1		liefert kontextbezogene Hilfestellungen
F2	*Breakpoints / Toggle*	Breakpoint an der Cursorposition setzen/ löschen
F3	*View / Module*	Auswahl eines Moduls
F4	*Run / Go to cursor*	Programm bis zur Cursorposition laufen lassen
F5	*Window / Zoom*	Zoomen des aktuellen Fensters
F6	*Window / Next Window*	zum nächsten Fenster springen
F7	*Run / Trace into*	einzelne Zeile oder Befehl ausführen
F8	*Run / Step over*	einzelne Zeile oder Befehl ausführen, Unterprogrammaufruf überspringen
F9	*Run / Run*	Programm laufen lassen
F10		Menüleiste aktivieren
[Alt][F1]	*Help / Previous topic*	letzten Hilfsbildschirm darstellen

Taste	Menübefehl	Funktion
[Alt][F2]	*Breakpoints/At*	Breakpoint auf eine bestimmte Adresse setzen
[Alt][F3]	*Window/Close*	aktuelles Fenster schließen
[Alt][F4]	*Run/Back trace*	Programmausführung rückwärts
[Alt][F5]	*Window/User screen*	aktuellen Ausgabebildschirm anschauen
[Alt][F6]	*Window/Undo close*	zuletzt geschlossenes Fenster wieder öffnen
[Alt][F7]	*Run/Instruction trace*	Einzelbefehl ausführen
[Alt][F8]	*Run/Until return*	Programm laufen lassen
[Alt][F9]	*Run/Execute to*	Programm bis zu einer bestimmten Adresse laufen lassen
[Alt][F10]		lokales Menü des aktuellen Fensters aktivieren
[Alt][1]-[9]		Umschalten zu einem der mit 1-9 numerierten Fenster
[Alt]		
[Leertaste]		Aufruf des Menüs ≅ (System)
[Alt][B]		Aufruf des Menüs *Breakpoints*
[Alt][D]		Aufruf des Menüs *Data*
[Alt][F]		Aufruf des Menüs *File*
[Alt][H]		Aufruf des Menüs *Help*
[Alt][O]		Aufruf des Menüs *Options*
[Alt][R]		Aufruf des Menüs *Run*
[Alt][V]		Aufruf des Menüs *View*
[Alt][W]		Aufruf des Menüs *Window*
[Alt][X]	*File/Quit*	Turbo Debugger verlassen
[Alt][=]	*Options/Macros/ Create*	Tastaturmakro definieren
[Alt][-]	*Options/Macros/ Stop recording*	Makroaufnahme beenden
[Ctrl][F2]	*Run/Program reset*	Programm erneut laden
[Ctrl][F4]	*Data/Evaluate*	Ausdruck auswerten
[Ctrl][F5]	*Windows/Size/Move*	
[Ctrl][F7]	*Data/Watch*	Variable im *Watches*-Fenster aufnehmen
[Ctrl][F8]	*Breakpoints/Toggle*	Breakpoint umschalten
[Ctrl][F9]	*Run/Run*	Programm laufen lassen
[Ctrl][F10]		lokales Menü eines Fensters aufrufen
[Ctrl][→]		verschiebt die Anfangsadresse in einem Code-, Daten- oder Stack-Ausschnitt eines *CPU*-Fensters um ein Byte nach oben
[Ctrl][←]		verschiebt die Anfangsadresse in einem Code-, Daten- oder Stack-Ausschnitt eines *CPU*-Fensters um ein Byte nach unten
[Ctrl][A]		zum vorherigen Wort
[Ctrl][C]		einen Ausschnitt nach unten rollen
[Ctrl][D]		eine Spalte nach rechts
[Ctrl][E]		eine Zeile nach oben
[Ctrl][F]		zum nächsten Wort
[Ctrl][R]		einen Ausschnitt nach oben rollen
[Ctrl][S]		eine Spalte nach links
[Ctrl][X]		eine Zeile nach unten
[Shift][F1]	*Help/Index*	Aufruf des Hilfe-Index
[Shift][Tab]		versetzt den Cursor in den vorherigen Fensterausschnitt

Taste	Menübefehl	Funktion
[Shift][↑]		in die entsprechende
[Shift][↓]		Richtung bewegen
Esc		schließt ein *Inspector*-Fenster
Ins		Anfang einer Blockauswahl (hervorgehoben)
Tab	*Window / Next pane*	versetzt den Cursor in den nächsten Fensterausschnitt

Die Befehle in der Menüleiste

Die Menüleiste wird mit der Taste F10 aufgerufen. In die einzelnen Menüs können Sie auch direkt gehen, indem Sie entweder

- mit dem Cursor zur Menübezeichnung gehen und ↵ drücken
- oder den ersten Buchstaben der Menübezeichnung eingeben. Sie können das Menü mit Alt und dem ersten Buchstaben des gewünschten Menüs auch direkt öffnen.

Das Menü System

Restore standard	Standard-Anordnung der Fenster wiederherstellen
Repaint desktop	gesamten Bildschirm neu aufbauen
About	Informationen über Turbo Debugger

Das Menü File

Load	neues Programm zum Debuggen laden
Change dir	Verzeichnis oder Laufwerk wechseln
Get info	Informationen über das Programm darstellen
Symbol load	Symboltabelle getrennt laden
Quit	Rückkehr

Das Menü View

Breakpoints	Breakpoints im Fenster darstellen
Stack	Stack mit den Funktionsaufrufen darstellen
Log	Protokoll von Ereignissen und Daten im Fenster darstellen

Watches	Variablen zur Überwachung darstellen
Variables	Globale und lokale Variablen darstellen
Module	Quellcode darstellen
File	Dateien im ASCII-Format oder hexadezimal darstellen
CPU	CPU-Befehle, Daten und Stack darstellen
Dump	Daten hexadezimal darstellen
Registers	CPU-Register und Flags darstellen
Numeric processor	Koprozessor oder Emulator darstellen
Execution history	ausgeführte Programmschritte und aufgezeichnete Tastatureingaben für Ausführung rückwärts darstellen
Hierarchy	Objekt- oder Klassentypliste und Hierarchiebaum darstellen
Windows messages	eine Liste von Windows-Meldungen für ein oder mehrere Fenster in Ihrem Anwendungsprogramm anzeigen
Another	
Module	zusätzliches *Module*-Fenster anlegen
Dump	zusätzliches *Dump*-Fenster anlegen
File	zusätzliches *File*-Fenster anlegen

Das Menü Run

Run	Programm ohne anzuhalten laufen lassen
Go to cursor	Programm läuft bis zur aktuellen Cursorposition durch
Trace into	Eine Quellzeile oder einen Befehl ausführen
Step over	einen Befehl oder eine Quellzeile ausführen, wobei Unterprogrammaufrufe übersprungen werden
Execute to	Programm bis zur angegebenen Adresse laufen lassen
Until return	Programm solange laufen lassen, bis das Programm von der aktuellen Funktion zurückkehrt
Animate	Programm Befehl für Befehl verlangsamt ausführen
Back trace	rückwärtige Ausführung eines Programmschritts
Instruction trace	einzelnen Befehl ausführen
Arguments	Kommandozeilenargumente festlegen
Program reset	aktuelles Programm neu laden

Das Menü Breakpoints

Toggle	Breakpoint umschalten
At	Breakpoint an einer bestimmten Adresse setzen
Changed memory global	globalen Breakpoint für Veränderungen eines Speicherbereichs setzen

Expression true global	globalen Breakpoint für einen Ausdruck setzen
Delete all	alle Breakpoints löschen

Das Menü Data

Inspect	Datenobjekt untersuchen
Evaluate/Modify	Ausdruck auswerten
Add Watch	Variable im *Watches*-Fenster eintragen
Function return	Funktionsergebnis einer Routine untersuchen

Das Menü Options

Language	Syntax für Ausdrücke entsprechend dem Quellmodul wählen
Macros	
Create	Tastaturmakro definieren
Stop recording	Makroaufnahme beenden
Remove	Makro löschen
Delete all	alle Makros löschen
Display options	Bildschirmoptionen festlegen
Path for source	Pfad für Quelldateien
Save options	Optionen, Makros und Fenstereinstellung speichern
Restore options	Optionen laden

Das Fenster Window

Zoom	Fenster auf volle Bildschirmgröße oder auf Ausgangsgröße bringen
Next	aktiviert nacheinander die geöffneten Fenster
Next pane	zum nächsten Fensterausschnitt springen
Size/Move	Position und Größe eines Fensters ändern
Iconize/Restore	Fenster in Symbol verwandeln und umgekehrt
Close	Fenster schließen
Undo close	zuletzt geschlossenes Fenster öffnen
Dump pane to log	aktuellen Fensterausschnitt in Log-Fenster schreiben
User screen	Ausgabebildschirm betrachten

Numbered window list	Liste der geöffneten und nummerierten Fenster anzeigen
Window pick	Fenster aus der Liste offener Fenster auswählen

Das Menü Help

Index	Aufruf des Hilfe-Index
Previous topic	vorletzten Hilfe-Bildschirm
Help on help	Hilfestellungen zum Hilfesystem

Die Befehle in den lokalen Menüs

Sie können das lokale Menü eines Fensters mit [Alt][F10] aufrufen. Wenn die Tastenkürzel mit der [Ctrl]-Taste aktiviert sind, kann man die einzelnen Menübefehle auch mit [Ctrl] und dem ersten Buchstaben des Befehls aktivieren. (Die [Ctrl]-Tasten können im Installationsprogramm TDWINST ein- und ausgeschaltet werden.)

Jedes Fenster (*Breakpoints*, *CPU* usw.) und jeder Ausschnitt in einem Fenster hat ein eigenes lokales Menü. In den folgenden Abschnitten beschreiben wir das lokale Menü für jedes Fenster und jeden Fensterausschnitt.

Einige Fensterausschnitte haben Tastenkürzel für häufig verwendete Befehle in ihren lokalen Menüs. Im folgenden Abschnitt werden diese Spezialtasten vor den Menübefehlen für den entsprechenden Ausschnitt angegeben. In vielen Ausschnitten wird die [–]-Taste als Kurzform zum Untersuchen und Verändern des gerade hervorgehobenen Objekts verwendet. Die [Del]-Taste dient häufig als Kurzform für einen Löschbefehl in einem lokalen Menü. In einigen Ausschnitten kann man einfach Zahlen und Buchstaben eingeben, ohne daß man einen Befehl in einem lokalen Menü auswählen muß. In diesen Fällen erscheint sofort ein Dialogfenster für das entsprechende Datenobjekt.

Das lokale Menü des Breakpoints-Fensters

Das *Breakpoints*-Fenster hat zwei Ausschnitte: links den Listenausschnitt und rechts den Detailausschnitt. Nur der Listenausschnitt weist ein lokales Menü auf.

Set options	Breakpoint-Aktionen, Bedingungen, Durchlaufzähler festlegen und ein-/ausschalten
Add	Breakpoint einfügen
Global	globalen Breakpoint einfügen

Remove	hervorgehobenen Breakpoint löschen
Delete all	alle Breakpoints löschen
Inspect	Code untersuchen, an dem der Breakpoint liegt

Das Tastenkürzel für *Remove* lautet in diesem Fenster Del.

Die Menüs des CPU-Fensters

Das *CPU*-Fenster hat fünf Ausschnitte, die alle ein lokales Menü besitzen: Code-Ausschnitt, Datenausschnitt, Stack-Ausschnitt, Registerausschnitt und Statusausschnitt.

Das lokale Menü des Codeausschnitts

Goto	Code der neuen Adresse darstellen
Origin	Code von CS:IP darstellen
Follow	Code der Adresse eines JMP- oder CALL-Ziels darstellen
Caller	Code der aufrufenden Funktion darstellen
Previous	Code der letzten Adresse darstellen
Search	Befehle oder Bytes suchen
View source	Umschalten auf *Module*-Fenster
Mixed	Auswahl der gemischten Darstellung von Quelltext und disassemblierten Befehlen: *No/Yes/Both*
New CS:IP	CS:IP auf neue Adresse setzen
Assemble	Befehl an der Cursorposition assemblieren
I/O	Menü zur Ein-/Ausgabe auf Port aufrufen
In byte	Byte von Port einlesen
Out byte	Byte an Port ausgeben
Read word	Wort von Port lesen
Write word	Wort an Port ausgeben

Die Eingabe von Zeichen wird als Kurzform des *Assemble*-Befehls interpretiert.

Das lokale Menü des Datenausschnitts

Goto	Daten der neuen Adresse darstellen
Search	Strings oder Bytefolgen suchen
Next	Suchbefehl wiederholen
Change	Daten an der Cursorposition verändern
Follow	
Near code	Code-Ausschnitt auf ausgewählte NEAR-Adresse setzen
Far code	Code-Ausschnitt auf ausgewählte FAR-Adresse setzen
Offset to data	Datenausschnitt auf ausgewählte NEAR-Adresse setzen

Segment: Offset to data	Datenausschnitt auf ausgewählte FAR-Adresse setzen
Base segment:0 to Data	Datenausschnitt auf den Anfang des Segments setzen mit der Adresse unter dem Cursor
Previous	Daten der letzten Adresse darstellen
Display as	
Byte	Daten als Hexadezimal-Bytes darstellen
Word	Daten als Hexadezimal-Worte darstellen
Long	Daten als Hexadezimal-Doppelworte darstellen
Comp	Daten als 8-Byte-Integer darstellen
Float	Daten als 4-Byte-Fließkommazahl darstellen
Real	Daten als 6-Byte-Fließkommazahl darstellen
Double	Daten als 8-Byte-Fließkommazahl darstellen
Extended	Daten als 10-Byte-Fließkommazahl darstellen
Block	
Clear	Block löschen
Move	Block verschieben
Set	Block initialisieren
Read	Block von Datei in Speicher einlesen
Write	Block von Speicher auf Datei schreiben

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Statusausschnitts

Toggle	hervorgehobenes Flag setzen oder löschen

Die Eingabetaste ⏎ oder die Leertaste wird als Kurzform des Befehls interpretiert.

Das lokale Menü des Registerausschnitts

Increment	hervorgehobenes Register um 1 erhöhen
Decrement	hervorgehobenes Register um 1 erniedrigen
Zero	hervorgehobenes Register löschen
Change	Inhalt des hervorgehobenen Registers ändern
Registers 32-bit	Umschalten zwischen 16- und 32-Bit-Darstellung: *No/Yes*

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Stack-Ausschnitts

Goto	Stack an der neuen Adresse darstellen
Origin	Stack an SS:SP darstellen
Follow	Stack mit der Adresse des hervorgehobenen Objekts darstellen
Previous	Letzte Adresse darstellen

Change	Wert am Stack verändern

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das Fenster Dump

Das Dump-Fenster entspricht dem Datenausschnitt des *CPU*-Fensters. Das lokale Menü ist identisch zum lokalen Menü des Datenausschnitts.

Das Menü des Fensters Execution History

Das Fenster *Execution History* hat zwei Ausschnitte, von denen jeder ein lokales Menü beinhaltet: Den Ausschnitt *Instructions* (Befehle) und den Ausschnitt *Keystroke Recording* (Aufzeichnung der Tastenanschläge).

Das lokale Menü des Instructions-Ausschnitts

Im Ausschnitt *Instructions* werden bereits ausgeführte Befehle aufgelistet, damit Sie die Befehle anschauen oder rückgängig machen können.

Inspect	Bringt Sie zu dem hervorgehobenen Befehl.
Reverse Execute	Läßt das Programm bis zu dem hervorgehobenen Befehl rückwärts ablaufen.
Full History	Schaltet den Rückwärtslauf des Programm ein (*On*) oder aus (*Off*).

Das Menü des Fensters File

Das *File*-Fenster stellt einen Dateiinhalt hexadezimal oder als ASCII-Text dar.

Goto	Datei ab Zeilennummer oder Hex-Offset darstellen
Search	String oder Bytes suchen
Next	Suche wiederholen
Display as	ASCII/Hex: Darstellungsmodus umschalten
File	neue Datei darstellen

Die Eingabe von Zeichen wird als Kurzform des *Search*-Befehls interpretiert.

Das Menü des Log-Fensters

Das *Log*-Fenster zeigt die Meldungen, die ins Protokoll geschrieben werden.

Open log file	Meldungen im Log-Fenster in Datei schreiben
Close log file	Log-Datei schließen
Logging	Protokoll ein- oder ausschalten: *No/Yes*
Add comment	Kommentar in *Log*-Fenster einfügen
Erase log	Meldungen im *Log*-Fenster löschen
Display Windows info	Meldungen im Windows-Fenster anschauen, wobei Sie zwischen globalem und lokalem Heap oder der Modulliste wählen können

Die Eingabe von Zeichen wird als Kurzform des Befehls *Add comment* interpretiert.

Das Menü des Module-Fensters

Das *Module*-Fenster zeigt die Quelldatei des Programmmoduls.

Inspect	Inhalt der Variablen unter dem Cursor untersuchen
Watch	Variable unter dem Cursor in *Watches*-Fenster aufnehmen
Module	in ein anderes Modul wechseln
File	andere Datei darstellen
Previous	letztes Modul an der entsprechenden Position darstellen
Line	Zeilennummer im Modul darstellen
Search	String suchen
Next	Suche wiederholen
Origin	Aktuelle Programmposition darstellen
Goto	Modul an neuer Position darstellen

Die Eingabe von Zeichen wird als Kurzform des *Goto*-Befehls interpretiert.

Die Menüs des Fensters Windows Messages

Das Fenster *Windows Messages* hat drei Aussschnitte: den Ausschnitt *Window Selection,* den Ausschnitt *Message Class* und den Ausschnitt *Messages*.

Das lokale Menü des Ausschnitts Windows Selection

Add	fügt einen Fensternamen oder Handlewert hinzu
Remove	löscht das selektierte Fenster
Delete all	löscht alle Fensterauswahlen

Die Eingabe von Zeichen wird als Kurzform des *Add*-Befehls in diesem Ausschnitt interpretiert.

Die Del-Taste und die Tastenkombination [Ctrl][Y] sind Abkürzungen für den Befehl *Remove*.

Das lokale Menü des Ausschnitts Windows Selection

Add	fügt eine Meldungsklasse oder eine einzelne Meldung hinzu
Remove	löscht die selektierte Meldungsklasse oder Einzelmeldung
Delete all	löscht alle Meldungsklassen- oder Einzelmeldungsauswahlen

Die Eingabe von Zeichen wird als Kurzform des *Add*-Befehls in diesem Ausschnitt interpretiert.

Die [Del]-Taste und die Tastenkombination [Ctrl][Y] sind Abkürzungen für den Befehl *Remove*.

Das lokale Menü des Ausschnitts Messages

Send log to window	Sendet alle empfangenen Meldungen an das *Log*-Fenster, so daß sie in einer Log-Datei gespeichert werden können.
Erase log	Löscht alle Meldungen in diesem Ausschnitt.

Die Menüs des Fensters Numeric processor

Das Fenster des Numerikprozessors hat drei Ausschnitte: den Registerausschnitt, den Statusausschnitt und den Kontrollausschnitt.

Das lokale Menü des Registerausschnitts

Die Tasten [Z], [E] und [C] sind Abkürzungen zu den lokalen Menübefehlen in diesem Ausschnitt.

Zero	löscht das hervorgehobene Register
Empty	setzt das hervorgehobene Register auf leer
Change	verändert den Inhalt des hervorgehobenen Registers

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Statusauschnitts

Toggle	Flag-Werte durchschalten

Die Eingabetaste [↵] ist eine Abkürzung des lokalen Menübefehls in diesem Ausschnitt.

Das lokale Menü des Kontrollausschnitts

Toggle	Flag-Werte durchschalten

Die Eingabetaste [↵] ist eine Abkürzung des lokalen Menübefehls in diesem Ausschnitt.

Das Fenster Hierarchy

Das *Hierarchy*-Fenster hat zwei Ausschnitte: eine Liste mit den Objekttypen/Klassen und einen Hierarchiebaum.

Der Auschnitt mit den Objekttypen/Klassen

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen
Tree	Wechsel in den Hierarchiebaum

Der Ausschnitt mit dem Hierarchiebaum

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen
Parents	Wechsel in den Vorfahrenbaum, falls die Mehrfachverbung verwendet wird

Der Ausschnitt mit dem Vorfahrenbaum

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen

Das Menü Registers

Das Fenster *Registers* entspricht den Register- und Statusausschnitten des *CPU*-Fensters. Die lokalen Menüs sind identisch zum lokalen Menü des Registerausschnitts und Statusausschnitts.

Das Menü des Stack-Fensters

Das *Stack*-Fenster zeigt die gerade aktivierten Funktionen.

Inspect	Quellcode der hervorgehobenen Funktion darstellen
Locals	Argumente der Funktion anzeigen

Die Eingabetaste ⏎ ist eine Abkürzung zum lokalen Menübefehl *Inspect*.

Die Menüs des Fensters Variables

Das *Variables*-Fenster enthält zwei Ausschnitte, die beide ein lokales Menü besitzen: den Globalauschnitt und den Lokalausschnitt

Das lokale Menü des Globalausschnitts

Inspect	zeigt den Inhalt des hervorgehobenen Symbols
Change	verändert den Inhalt des hervorgehobenen Symbols
Watches	Fügt das hervorgehobene Symbol im Fenster *Watches* hinzu

Die Eingabetaste ⏎ ist eine Abkürzung zum lokalen Menübefehl *Inspect*.

Das lokale Menü des Lokalausschnitts

Inspect	zeigt den Inhalt des hervorgehobenen Symbols
Change	verändert den Inhalt des hervorgehobenen Symbols
Watches	Fügt das hervorgehobene Symbol im Fenster *Watches* hinzu
Show	Wechselt das Modul oder zeigt nur statische Variablen, nur Auto-Variablen oder beides.?? (static, auto, both, module) ??

Die Eingabetaste ⏎ ist eine Abkürzung zum lokalen Menübefehl *Inspect*.

Das Menü des Fensters Watches

Das *Watches*-Fenster hat einen einzigen Ausschnitt, der die Namen und die Werte der Variablen zeigt, die Sie beobachten.

Watch	Variable einfügen
Edit	Variablennamen editieren
Remove	hervorgehobene Variable löschen
Delete all	alle *Watches*-Variablen löschen
Inspect	Inhalt der hervorgehobenen Variablen untersuchen
Change	Inhalt der hervorgehobenen Variablen verändern

Die folgenden Tasten sind Abkürzungen zu den lokalen Menübefehlen in diesem Fenster.

Taste	Funktion
irgendein Zeichen	*Watch*
⏎	*Edit*
Del	*Remove*

Das Menü des Inspector-Fensters

Ein *Inspector*-Fenster zeigt den Inhalt eines Datenobjekts.

Range	wählt Array-Elemente für die Untersuchung aus
Change	verändert den Wert des hervorgehobenen Objekts

Inspect	öffnet ein neues *Inspector*-Fenster für das hervorgehobene Datenobjekt
Descend	erweitert das hervorgehobene Datenobjekt in diesem *Inspector*-Fenster
New expression	untersucht einen neuen Ausdruck in diesem *Inspector*-Fenster
Type cast	Datenvorgabe für das ausgewählte Datenobjekt

Das Inspector-Fenster für Objekttypen/Klassen

Das *Inspector*-Fenster für Objekttypen/Klassen hat zwei Ausschnitte, die den Inhalt (Datenfelder oder -elemente und Methoden oder Elementfunktionen) einer Klasse oder eines Objekts anzeigen. Die lokalen Menüs, die in beiden Ausschnitten gleich sind, unterscheiden sich sehr stark von den üblichen *Inspector*-Fenstern.

Inspect	*Inspector*-Fenster für den ausgewählten Typ öffnen
Show inherited	Umschalten zwischen der Darstellung des Inhalts des Objekts/Klasse inklusive aller vererbten Elemente und dem Inhalt des aktuellen Objekts oder Klasse
Hierarchy	Rückkehr zum *Hierarchy*-Fenster

Das Inspector-Fenster für Objekt-/Klasseninstanzen

Die *Inspector*-Fenster für Instanzen von Objeten oder Klassen enthält drei Ausschnitte, von denen nur die ersten beiden ein lokales Menü besitzen. (Der dritte gibt nur den Objekttyp oder die Klasse an, zu der die Instanz gehört). Die beiden lokalen Menüs sind gleich und enthalten folgenden Befehle:

Range	Array-Elemente für die Untersuchung auswählen
Change	Wert des ausgewählten Datenojekts ändern
Methods	Umschalten, ob Methoden oder Elementfunktionen im mittleren Ausschnitt zusammengefaßt werden
Show inherited	Umschalten zwischen der Darstellung des Inhalts des Objekts/Klasse inklusive aller vererbten Elemente und dem Inhalt des aktuellen Objekts oder Klasse
Inspect	*Inspector*-Fenster für das aktuelle Element öffnen
Descend	ausgewähltes Datenobjekt in einem *Inspector*-Fenster erweitern
New expression	untersucht einen neuen Ausdruck in diesem *Inspector*-Fenster
Type cast	Datenvorgabe für das ausgewählte Datenobjekt
Hierarchy	Rückkehr zum *Hierarchy*-Fenster

Textausschnitte

Die Bezeichnung Textausschnitt ist eine allgemeine Bezeichnung für alle Ausschnitte, in denen der Inhalt einer Textdatei dargestellt wird. Der blinkende Cursor gibt die aktuelle Position in der Datei an. In der folgenden Tabelle finden Sie alle Befehle:

Tabelle 14.2 Tastaturbefehle in Textausschnitten

Taste	Funktion
Ins	markiert einen Textblock
↑	bewegt den Cursor um eine Zeile nach oben
↓	bewegt den Cursor um eine Zeile nach unten
→	bewegt den Cursor um eine Spalte nach rechts
←	bewegt den Cursor um eine Spalte nach links
[Ctrl][→]	bewegt den Cursor zum nächsten Wort
[Ctrl][←]	bewegt den Cursor zum vorherigen Wort
[Ctrl][End]	bewegt den Cursor in die letzte Zeile des Ausschnitts
[Ctrl][PgUp]	bewegt den Cursor in die erste Zeile der Datei
[Ctrl][PgDn]	bewegt den Cursor in die letzte Zeile der Datei

Wenn Sie die Ctrl-Tastenkürzel nicht verwenden, dann können Sie in einem Textausschnitt auch die Wordstar-Befehle verwenden.

Listen-ausschnitte

Die Bezeichnung Listenausschnitt ist eine allgemeine Bezeichnung für alle Ausschnitte, in denen Listen von Datenobjekten dargestellt werden. Der Auswahlbalken gibt die aktuelle Position in der Liste an. In der folgenden Tabelle finden Sie alle Befehle:

Tabelle 14.3 Tastaturbefehle in Listenausschnitten

Taste	Funktion
↑	bewegt den Auswahlbalken um eine Zeile nach oben
↓	bewegt den Auswahlbalken um eine Zeile nach unten
→	rollt den Ausschnitt nach rechts
←	rollt den Ausschnitt nach links
Home	bewegt den Auswahlbalken an den Anfang der Zeile
End	bewegt den Auswahlbalken an das Ende der Zeile
[PgUp]	rollt die Darstellung um einen Ausschnitt nach oben
PgDn	rollt die Darstellung um einen Ausschnitt nach unten
[Ctrl][Home]	bewegt den Auswahlbalken in die erste Zeile des Ausschnitts
[Ctrl][End]	bewegt den Auswahlbalken in die letzte Zeile des Ausschnitts

Taste	Funktion
[Ctrl][PgUp]	bewegt den Auswahlbalken zum ersten Objekt in der Liste
[Ctrl][PgDn]	bewegt den Auswahlbalken zum letzten Objekt in der Liste
Backspace	nimmt ein Zeichen bei der schrittweisen Suche zurück
Buchstabe	schrittweises Suchen (Auswahl durch Eintippen)

Wenn Sie die Ctrl-Tastenkürzel nicht verwenden, dann können Sie in einem Listenausschnitt auch die Wordstar-Befehle verwenden.

Funktionstasten in Dialogfenstern und Eingabefeldern

Die folgende Tabelle zeigt, welche Funktionstasten innerhalb eines Dialogfensters zur Verfügung stehen.

Tabelle 14.4 Tastaturbefehle in Dialogfenstern

Taste	Funktion
↑	bewegt den Cursor um einen Eintrag nach oben
↓	bewegt den Cursor um einen History-Eintrag nach unten
→	bewegt den Cursor um ein Zeichen nach rechts
←	bewegt den Cursor um ein Zeichen nach links
[Ctrl][→]	bewegt den Cursor zum nächten Wort
[Ctrl][←]	bewegt den Cursor zum vorherigen Wort
Home	bewegt den Cursor an den Anfang der Zeile
End	bewegt den Cursor an das Ende der Zeile
[PgUp]	rollt die Darstellung um einen Ausschnitt nach oben
PgDn	rollt die Darstellung um einen Ausschnitt nach unten
[Ctrl][Home]	bewegt den Cursor in die erste Zeile des Ausschnitts
[Ctrl][End]	bewegt den Cursor in die letzte Zeile des Ausschnitts
[Ctrl][PgUp]	bewegt den Cursor in die erste Zeile der Datei
[Ctrl][PgDn]	bewegt den Cursor in die letzte Zeile der Datei
Backspace	löscht das Zeichen vor dem Cursor
↵	akzeptiert die Eingabe und macht weiter
Del	löscht das Zeichen unter dem Cursor
Esc	bricht den Dialog ab und kehrt zum Menü zurück
[Ctrl][N]	vervollständigt einen teilweise eingegebenen Namen in einem Eingabefeld

Befehle zum Verändern der Fensteranordnung

Die folgende Tabelle zeigt die Tasten, mit den Sie ein Fenster verschieben oder seine Größe verändern können.

Tabelle 14.5 Tastaturbefehle zum Verändern der Fensteranordnung

Taste	Funktion
[Ctrl][F5]	schaltet die Fensterpositionierung um
[↑]	bewegt das Fenster um eine Zeile nach oben
[↓]	bewegt das Fenster um eine Zeile nach unten
[→]	bewegt das Fenster um eine Spalte nach rechts
[←]	bewegt das Fenster um eine Spalte nach links
[Shift][↑]	verkleinert das Fenster von unten her
[Shift][↓]	vergrößert das Fenster nach unten
[Shift][→]	verkleinert das Fenster von rechts her
[Shift][←]	vergrößert das Fenster nach rechts
[Home]	bewegt das Fenster auf die linke Seite des Bildschirms
[End]	bewegt das Fenster auf die rechte Seite des Bildschirms
[PgUp]	bewegt das Fenster in die oberste Zeile des Bildschirms
[PgDn]	bewegt das Fenster in die unterste Zeile des Bildschirms
[↵]	akzeptiert die aktuelle Position
[Esc]	bricht die Fensterpositionierung ab

Jokerzeichen

Sie können die Jokerzeichen in zwei Fällen verwenden:

- wenn Sie einen Dateinamen eingeben, um eine Datei zu laden oder zu untersuchen;
- wenn Sie einen String in einem Textausschnitt suchen.

Das Fragezeichen (?) steht für ein beliebiges Zeichen im Suchstring. Das Sternchen (*) wird stellvertretend für 0 oder mehrere Zeichen im Suchstring angegeben.

Der vollständige Menübaum

In Abbildung 14.1 ist die komplette Struktur der Turbo-Debugger-Menüs dargestellt.

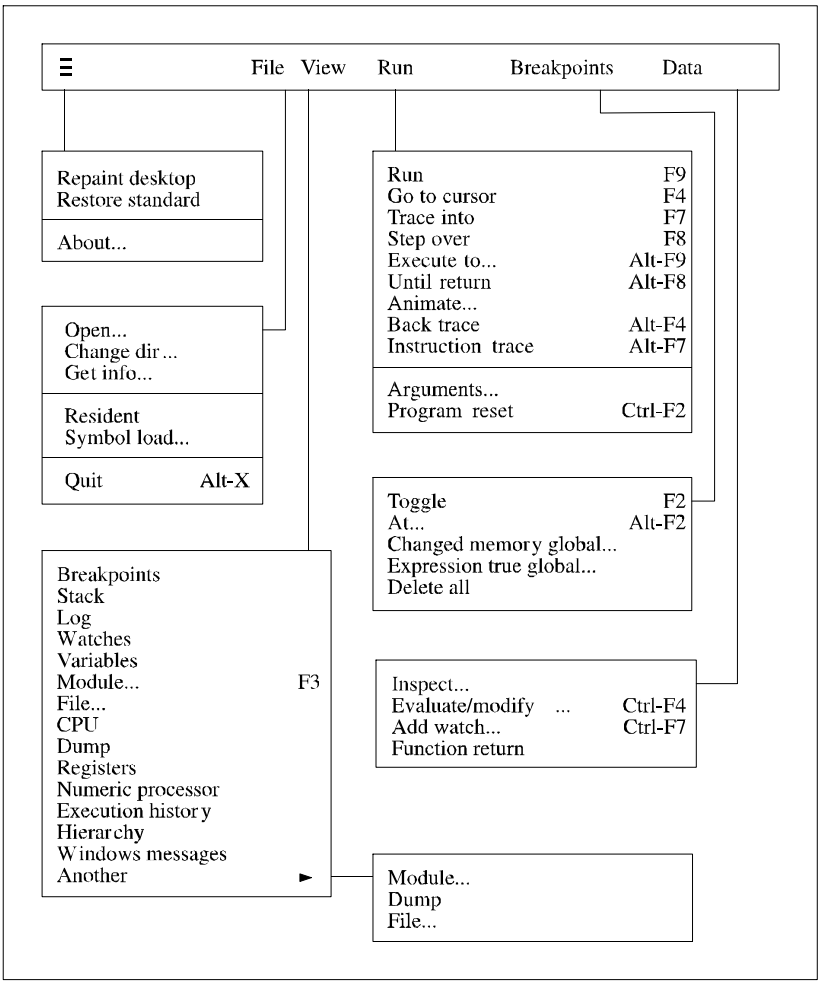


Abbildung 14.1 Der Menübaum von Turbo Debugger

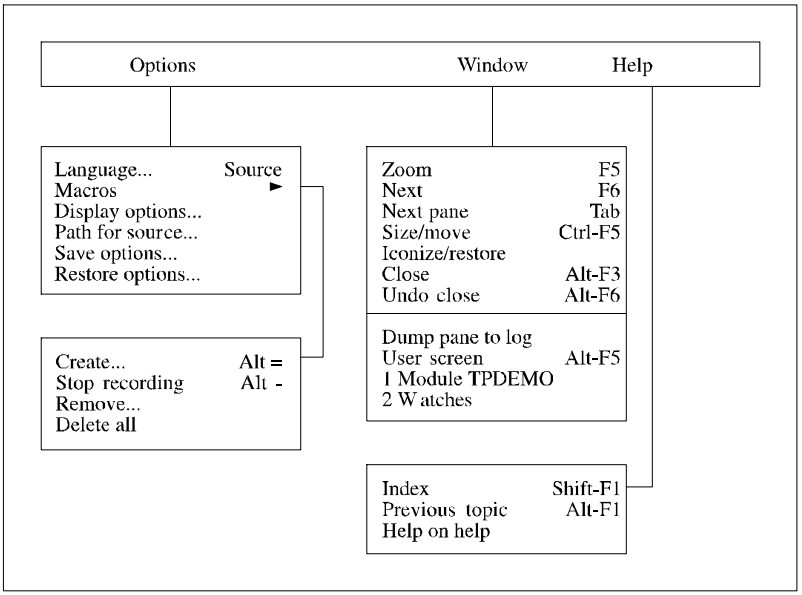


Abbildung 14.2 Der Menübaum von Turbo Debugger

15. Ein Standard-Pascal-Programm debuggen

Das Debuggen von Programmen ist, wie die anderen Phasen der Programmerstellung, zum Teil eine Wissenschaft, zum Teil aber auch eine Kunst. Es gibt bestimmte Prozeduren, um einen Fehler aufzuspüren, während man in der gleichen Zeit mit etwas Intuition einen umständlichen Weg verkürzen kann.

Je mehr Programme Sie debuggen, desto schneller werden Sie die Ursachen für Probleme in Ihrem Programm finden. Sie werden Methoden kennenlernen, die Ihnen dabei entgegenkommen, und Sie werden lernen, wie man Programmiermethoden ändert, die Ihnen immer wieder Probleme bereiten.

In diesem Kapitel besprechen wir einige unterschiedliche Wege beim Debuggen, unterhalten uns über die verschiedenen Arten von Fehlern, die vielleicht in Ihren Programmen auftauchen können und schlagen einige Methoden vor, wie Sie Ihr Programm testen können, damit es zuverlässig läuft und auch in Zukunft laufen wird.

Wir wollen einmal sehen, wo anzufangen ist, wenn Sie ein Programm haben, das nicht richtig arbeitet.

Wenn's nicht läuft

Zuallererst: Keine Panik! Auch der erfahrenste Programmierer schreibt selten ein Programm, das beim ersten Mal läuft. Um nicht allzu viel Zeit zu verlieren, muß man der Versuchung widerstehen können, wahllos herumzuraten, wo der Fehler liegen könnte.

Gehen Sie von einer Reihe von Annahmen aus, und testen Sie jede einzeln durch. Sie könnten zum Beispiel folgende Hypothese aufstellen: »Der Fehler muß vor dem Aufruf der Funktion xyz auftreten.« Anschließend prüfen Sie diese Hypothese, indem Sie Ihr Programm direkt vor dem Funktionsaufruf von xyz anhalten und nachschauen, ob sich bis dahin ein Problem ergeben hat. Wenn Sie an diesem Punkt ein Problem entdecken, können Sie in einer neue Hypothese annehmen, daß das Problem noch viel früher in Ihrem Programm auftritt.

Wenn aber bis zum Funktionsaufruf von xyz alles in Ordnung ist, dann war Ihr erster Ausgangspunkt falsch. Sie müssen nun Ihre Hypothese wie folgt verändern: »Der Fehler tritt nach dem Funktionsaufruf von xyz auf.« Indem Sie

eine Reihe von Tests wie diesen ausführen, werden Sie sehr bald den Programmbereich finden, in dem das Problem verursacht wird.

Das ist alles schön und gut, werden Sie vielleicht jetzt sagen. Wie aber kann ich feststellen, daß das Programm korrekt arbeitet, wenn ich das Programm anhalte? Eine der besten Methoden, das Verhalten eines Programms zu überprüfen, ist die Untersuchung der Variablen und Datenobjekte im Programm. Wenn Sie eine Routine haben, die ein Array löscht, dann können Sie die Operation überprüfen, indem Sie das Programm nach der Ausführung der Funktion anhalten und dann überprüfen, ob jedes Element des Arrays auch tatsächlich gelöscht wurde.

Der persönliche Stil beim Debuggen

Jeder hat seinen eigenen Programmierstil und entwickelt auch beim Debuggen seinen eigenen Stil. Die Vorschläge zum Debuggen, die wir Ihnen hier machen, sind lediglich ein Ausgangspunkt, von dem aus Sie Ihre eigenen Methoden entwickeln können.

Oft bestimmt der Verwendungszweck, wie Sie mit dem Debuggen anfangen. Einige Programme werden nur von Ihnen selbst eingesetzt oder nur ein- oder zweimal für eine bestimmte Aufgabe verwendet. Bei diesen Programmen ist ein vollständiger Test aller Komponenten wahrscheinlich Zeitverschwendung. Dies ist insbesondere dann der Fall, wenn Sie die Richtigkeit des Programms anhand der Ausgaben überprüfen können. Ein Programm, das an andere Leute verteilt werden soll, oder dessen Richtigkeit sehr schwer nachzuvollziehen ist, werden Sie sehr viel intensiver austesten.

Einfach mal laufen lassen

Bei einfachen Programmen oder Gelegenheitsprogrammen ist es meistens am besten, wenn Sie es einfach starten und sehen, was passiert. Wenn Ihr Testfall zu Problemen führt, können Sie das Programm mit den einfachsten Eingaben starten und dann die Ausgabe wieder überprüfen. Sie können dann weitergehen und kompliziertere Eingaben überprüfen, bis die Ausgabe falsch wird. Damit haben Sie sofort einen guten Überblick, wie viel oder wie wenig in Ihrem Programm falsch ist.

Schrittweise testen

Wenn Sie ganz sicher sein wollen, daß Ihr Programm korrekt ist, müssen Sie sowohl die einzelnen Routinen überprüfen als auch einige Programmläufe mit

Testdatensätzen durchführen. Sie haben dabei viele Möglichkeiten: Sie können jede Routine sofort beim Schreiben überprüfen, indem Sie ein Testprogramm für die Routine mit Testdaten erstellen. Man kann aber auch den Debugger beim fertigen Programm verwenden, um jede einzelne Routine durchzugehen.

Mögliche Fehler

Die Fehler in Ihren Programmen lassen sich in zwei große Kategorien einteilen: diejenigen, die typisch für die verwendete Sprache sind, und diejenigen, die in allen Programmiersprachen und Programmierumgebungen auftreten.

Wenn Sie beim Debuggen Ihrer Programme bewußt vorgehen, werden Sie sowohl die sprachspezifischen Konstrukte, mit denen Sie Probleme haben, als auch Ihre allgemeinen Programmierfehler erkennen. Mit diesem Wissen können Sie versuchen, diese Fehler in Zukunft zu vermeiden. Sie haben außerdem einen guten Ausgangspunkt zur Fehlersuche in zukünftigen Programmen.

Es ist wichtig, daß Sie jeden Fehler als einen Spezialfall einer allgemeinen Sorte von Fehlern verstehen, damit Sie sich auf diese Weise bei der Erstellung fehlerfreier Programme verbessern können. Es ist allemal besser, wenn Sie fehlerfreien Code schreiben, als wenn Sie zu einem Experten in der Fehlersuche werden.

Allgemeine Fehler

Die folgenden Beispiele decken nur sehr oberflächlich die Vielzahl verschiedener Fehlermöglichkeiten in einem Programm ab.

Versteckte Seiteneffekte

Wenn Sie sorglos globale Variablen in Prozeduren verwenden, kann deren Aufruf manchmal seltsame Ergebnisse in Variablen oder Datenstrukturen hinterlassen:

```
program Buffers;

uses WinCrt, Strings;

var
  WorkBuf,
  AllCaps : PCHar;

procedure Convert(S: PChar);
var
  I, Len : Integer;
begin
  Len := StrLen(S);
```

```
    StrCopy(WorkBuf, S);
    for I := 1 to Len do
      ;   { ... }
  end;

  begin
    WorkBuf := 'all done';
    AllCaps := 'SNAFU';
    Convert(AllCaps);
    Writeln(WorkBuf);
  end.
```

In diesem Fall wäre es richtig gewesen, wenn sich die rozedur ihren eigenen Arbeitspuffer angelegt hätte, statt auf WorkBuf zurückzugreifen.

Von initialisierten Daten darf man nicht ausgehen

Manchmal geht man davon aus, daß eine andere Routine eine Variable bereits mit einem bestimmten Wert belegt hat:

```
uses Strings;

var
  WorkBuf: PChar;

procedure AddWorkString(S: PChar);
begin
  StrCopy(WorkBuf, S);   { Aua! }
end;
```

Sicherer wäre die Prozedur, wenn sie folgende Anweisung enthalten würde:

```
if (WorkBuf = nil) then New(WorkBuf);
```

Aufräumen vergessen

Diese Sorte von Fehler kann sehr lange brauchen, bis das Programm abstürzt, weil der Heap ausgegangen ist:

```
function CrunchString(S: PChar): PChar;
var
 Work: PChar;
begin
 New(Work);
 StrCopy(S, Work);
 { ... }
 CrunchString := Work;    { Work immer noch reserviert... }
end;                      { ...und hier auch noch }
```

Kleinigkeit mit großer Wirkung

Bei dieser Art von Fehlern wird eine Konstante um den Wert 1 zuviel oder zuwenig angegeben:

```
i := 1;
```

```
while i < 10 do
begin
  { ... }
end;
```

In diesem Fall sieht man die Zahlen 1 und 10 und meint, daß die Schleife von eins bis zehn geht. Damit das Programmstück läuft, muß man den Operator < durch <= austauschen.

Typische Fehler in Pascal

Aufgrund der strengen Typ- und Fehlerkontrolle in Pascal gibt es nur sehr wenige Fehler, die für diese Programmiersprache typisch wären. Da Sie jedoch in Turbo Pascal viele der Kontrollmechanismen ausschalten können, können sich Fehler einschleichen, die Sie sonst nicht erhalten würden. Außerdem gibt es auch in Standard-Pascal noch einige Fehlermöglichkeiten.

Nicht initialisierte Variablen

Turbo Pascal initialisiert Variablen nicht automatisch. Das müssen Sie schon selbst machen und zwar entweder mit Zuweisungen oder durch die Deklaration als typisierte Konstanten. Betrachten Sie folgendes Programm:

```
program Test;

uses WinCrt;

var
  I,J,Anzahl: Integer ;
begin
  for I := 1 to Anzahl do
    begin
      J := I*I ;
      Writeln ( I:2, ' ', J:4 )
    end
end.
```

Die Variable Anzahl enthält irgendeinen Zufallswert, der zum Zeitpunkt des Funktionsaufrufs im Speicher stand, so daß Sie nicht wissen können, wie oft die Schleife wiederholt wird.

Außerdem werden alle Variablen innerhalb von Prozeduren oder Funktionen bei jedem Funktionsaufruf neu angelegt und wieder zerstört, wenn Sie die Funktion verlassen. Sie dürfen sich bei diesen Variablen nicht darauf verlassen, daß die Werte zwischen zwei Funktionsaufrufen erhalten bleiben.

Baumelnde Zeiger

Es gibt drei allgemeine Fehler mit Zeigern. Erstens dürfen sie nicht verwendet werden, wenn sie keinen Wert (NIL oder einen anderen Wert) zugewiesen bekommen haben. Wie alle anderen Variablen auch, werden Zeiger bei der

Deklaration nicht automatisch initialisiert. Sie sollten so schnell wie möglich einen Anfangswert erhalten (indem man sie in einem *New*-Aufruf übergibt oder nil zuweist).

Zweitens darf man einen nil-Zeiger nicht referenzieren. Man kann also nicht auf den Datentyp oder die Struktur zugreifen, auf die dieser Zeiger zeigt, wenn der Zeiger den Wert nil enthält. Nehmen wir zum Beispiel einmal an, daß eine einfach verkettete Liste von Records vorliegt, in der Sie einen Record mit einem bestimmten Wert suchen. Der Code kann wie folgt aussehen:

```
function KnotenFinden(Kopf: KnotenZgr; Schluessel: Integer):
                      KnotenZgr;
var
  Temp: KnotenZgr;
begin
  Temp := Kopf ;
  while ( Temp^.Schluessel <> Wert ) and ( Temp <> nil) do
    Temp := Temp^.Naechster;
  KnotenFinden := Temp
end; { von Funktion KnotenFinden }
```

Wenn Wert mit keinem der Schluessel-Felder in den Knoten der verketteten Liste übereinstimmt, wird in diesem Codeabschnitt versucht, Temp^. Schluessel auszuwerten, obwohl Temp den Wert nil hat, so daß das Verhalten der Routine nicht definiert ist. Wie lautet die Lösung? Die while-Bedingung muß folgendermaßen formuliert werden:

```
while (Temp <> nil) and (Temp^.Schluessel <> Wert)
```

Die Auswertung Boolescher Ausdrücke muß mit der Turbo-Pascal-Option {$B-} oder dem Befehl *Options / Compiler / Short Circuit* kurzgeschlossen werden. Auf diese Weise wird der zweite Ausdruck niemals ausgewertet, wenn Temp den Wert nil hat.

Sie dürfen auch niemals davon ausgehen, daß ein Zeiger den Wert nil hat, nur weil der Zeiger in den Funktionen *Dispose* oder *FreeMem* angegeben wurde. Der Zeiger hat immer noch denselben Wert, aber er zeigt jetzt auf einen Speicherbereich, der von anderen dynamischen Daten verwendet werden kann. Sie sollten einen Zeiger explizit auf nil setzen, wenn Sie den entsprechenden Datenbereich freigegeben haben.

Gültigkeitsbereich

In Pascal kann man Prozeduren und Funktionen sehr tief verschachteln, und alle Prozeduren und Funktionen haben ihre eigenen Deklarationen. Betrachten Sie das folgende Programm

```
program Verwechselt ;
uses WinCrt;

var
  A, B, T: Integer ;

procedure Swap(var A, B: Integer);
```

```
var
  T: Integer;
begin
  Writeln('2: A,B,T = ', A:3,B:3,' ',T);
  T := A;
  A := B;
  B := T;
  Writeln('3: A,B,T = ', A:3,B:3,' ',T);
end; { von Prozedur Swap }

begin { des Programms Verwechselt }
  A := 10 ; B := 20 ; T := 30;
  Writeln('1: A,B,T = ', A:3,B:3,' ',T);
  Swap( B, A );
  Writeln('4: A,B,T = ', A:3,B:3,' ',T);
end. { des Programms Verwechselt }
```

Was wird das Programm ausgeben? Es wird etwa folgendes sein:

```
1: A,B,T = 10 20 30
2: A,B,T = 20 10 22161
3: A,B,T = 10 20 20
4: A,B,T = 20 10 30
```

Hier sind zwei Versionen von `A, B` und `T` verwendet worden. Die globalen Versionen werden im Hauptteil des Programms verwendet, während `Swap` seine eigenen Variablen verwendet, nämlich die formalen Parameter `A` und `B` sowie die lokale Variable `T`. Zur weiteren Verwirrung wurde auch noch `Swap(B, A)` aufgerufen, so daß dem formalen Parameter `A` eigentlich die globale Variable `B` entspricht, und umgekehrt. Es gibt natürlich auch keinerlei Beziehung zwischen der lokalen und der globalen Version von `T`.

Es lag hier eigentlich kein richtiger »Fehler« vor, aber es entstehen viele Probleme, wenn Sie meinen, daß Sie etwas verändern, was Sie in Wahrheit aber gar nicht tun. Zum Beispiel wird die Variable `T` im Hauptprogramm überhaupt nicht verändert, auch wenn Sie vielleicht davon ausgegangen sind. Das ist im Grunde genommen das Gegenteil der versteckten Seiteneffekte.

Wenn Sie auch noch folgende Record-Definition gemacht hätten, dann wäre die Verwirrung perfekt:

```
type
  RecTyp = record
             A, B: Integer;
           end;

var
  A, B: Integer;
  Rec : RecTyp;
```

Innerhalb einer with-Anweisung würden sich `A` und `B` auf die Felder des Records beziehen und nicht auf die Variablen.

Überflüssige Strichpunkte

Wie in C, kann man auch in Pascal eine »leere« Anweisung angeben (die nur aus einem Strichpunkt besteht). An der falschen Stelle kann man damit allerhand Probleme verursachen. Betrachten Sie das folgende Programm:

```
program Test;
uses WinCrt;
var
  I, J: Integer;
begin
  for I := 1 to 20 do;
    begin
      J := I * I;
      Writeln(I:2, ' ', J:4);
    end;
  Writeln('Alles fertig');
end.
```

Die Ausgabe dieses Programms enthält nicht die ersten 20 Integer-Werte und deren Quadraten, sondern nur

```
20 400
Alles fertig!
```

Das kommt daher, weil die Anweisung `for I := 1 to 20 do ;` mit einem Strichpunkt abgeschlossen wurde. Deshalb wurde die leere Anweisung 20mal ausgeführt wurde. Anschließend werden die Anweisungen im `begin..end`-Block ausgeführt und schließlich die `Writeln`-Anweisung. Um das zu beheben, müssen Sie den Strichpunkt hinter `do` entfernen.

Funktionsergebnis nicht definiert

Wenn Sie eine Funktion schreiben, müssen Sie darauf achten, daß der Funktionsname einen Wert erhält, bevor die Funktion verlassen wird. Betrachten Sie folgenden Codeabschnitt:

```
const
  NLMax = 100;
type
  NumList = array[1..NLMax] of Integer;
  ...
function FindMax (Liste: NumList; Anzahl: Integer): integer;
var
  I, Max: Integer;
begin
  Max := Liste[1];
  for I := 2 to Anzahl do
    if Liste[I] > Max then
      begin
        Max := Liste[I];
        FindMax := Max
```

```
      end
end; { von Funktion FindMax }
```

Diese Funktion arbeitet prima, solange der größte Wert der Liste nicht in Liste[1] steckt. In diesem Fall erhält FindMax niemals einen Wert. Die korrekte Form dieser Funktion würde so aussehen:

```
begin
  Max := Liste[1];
  for I := 2 to Anzahl do
    if Liste[I] > Max then
      Max := Liste[I];
    FindMax := Max
end; { von Funktion FindMax }
```

Wort- oder Byte-Variablen dekrementieren

Passen Sie darauf auf, daß Sie keinen vorzeichenlosen Skalar (*Byte* oder *Word*) dekrementieren, während Sie auf Null prüfen. Der folgende Code erzeugt eine Endlosschleife:

```
var  w: Word;
begin
  w := 5;
  while w >= 0 do
    w := w - 1;
end.
```

Nach der fünften Iteration wird w gleich 0. Beim nächsten Mal wird es zu 65535 (da ein *Word* einen Wertebereich von 0 bis 65535 aufweist), was immer noch größer gleich Null ist. Sie sollten in solchen Fällen eine Integer-Zahl oder einen *LongInt* verwenden.

Grenz- oder Spezialfälle übersehen

Beachten Sie bitte, daß in beiden Versionen der Funktion `FindMax` im vorigen Abschnitt angenommen wurde, daß `Anzahl` größer gleich Eins ist. Es kann aber vielleicht auch den Fall geben, daß `Anzahl` gleich Null ist, die Liste also leer ist. Wenn Sie `FindMax` in so einem Fall aufrufen, wird es immer das zurückgeben, was in `Liste[1]` steht. Ähnlich ist es, wenn Anzahl größer `NLMax` ist. Dann erhält man einen Laufzeitfehler (wenn die Bereichsüberprüfung eingeschaltet ist) oder es werden Speicherbereiche durchsucht, die nicht in der Liste liegen.

In diesem Fall sind zwei Lösungen möglich. Die erste ist, daß man `FindMax` niemals aufruft, wenn Anzahl nicht im Bereich von `1..NLMax` liegt. Das ist ein ernstzunehmender Punkt. Einer der wichtigsten Bestandteile einer guten Software-Entwicklung ist die Definition der Anforderungen für den Aufruf einer bestimmten Funktion und die Überprüfung, ob die Bedingungen bei jedem Aufruf eingehalten werden.

Die andere Lösung wäre, daß man Anzahl überprüft und einen definierten Wert zurückgibt, wenn Anzahl nicht im Bereich von `1..NLMax` liegt. Sie könnten FindMax folgendermaßen umschreiben:

```
begin
  if (Anzahl < 1) or (Anzahl > NLMax) then
    Max := -32768
  else
    begin
      Max := Liste[1];
      for I := 2 to Anzahl do
        if Liste[I] > Max then
          Max := Liste[I];
    end;
  FindMax := Max
end; { von Funktion FindMax }
```

Das führt uns direkt zum nächsten typischen Pascal-Fehler: dem Bereichsfehler.

Bereichsfehler

In Turbo Pascal ist die Bereichsüberprüfung normalerweise ausgeschaltet. Das ergibt schnelleren und kompakteren Code, aber es ergeben sich damit auch gewisse Fehlerquellen, wie die Zuweisung von Variablen außerhalb des erlaubten Bereichs oder die Indizierung von nicht vorhandenen Elementen in einem Array, wie im letzten Beispiel.

Der erste Schritt, um solche Fehler zu finden, ist das Einschalten der Bereichsüberprüfung mit der Compiler-Option {$R}. Anschließend wird neu compiliert und nochmal gestartet. Wenn Sie wissen (oder ahnen), wo der Fehler liegt, können Sie diese Anweisung vor dem Abschnitt einschalten und hinter dem Abschnitt wieder mit {$R-} ausschalten, so daß die Bereichsüberprüfung nur für diesen Abschnitt gültig ist. Wenn ein Bereichsfehler auftritt, wird Ihr Programm mit einem Laufzeitfehler abgebrochen, und Turbo Pascal teilt Ihnen mit, wo der Fehler auftrat.

Eine Art von Bereichsfehler passiert sehr häufig, wenn Sie ein Array in einer while- oder repeat-Schleife indizieren. Sie suchen zum Beispiel ein Array-Element mit einem bestimmten Wert. Sie wollen die Suche abbrechen, wenn Sie das Element gefunden haben oder am Ende des Arrays angelangt sind. Wenn Sie das Element gefunden haben, wollen Sie den Index des Arrays zurückgeben. Ansonsten wird der Wert 0 zurückgeliefert. Ihr erster Versuch sieht vielleicht folgendermaßen aus:

```
function WertErmitteln(Liste: NumList; Anzahl, Wert: Integer): Inte-
ger;
var
  I := Integer;
begin
  WertErmitteln := 0;
  I := 1;
  while (I <= Anzahl) and (Liste[I] <> Wert) do
    Inc(I);
  if I <= Anzahl then
```

```
    WertErmitteln := I;
end; { von Funktion WertErmitteln }
```

Das ist alles sehr schön, aber die Funktion kann einen Laufzeitfehler ergeben, wenn Wert nicht in der Liste ist und Sie die normale Boolesche Auswertung vornehmen. Warum? Weil I beim letzten Test gleich Anzahl+1 ist. Wenn Anzahl = NLMax, dann befindet man sich außerhalb der Grenzen von Liste.

Es gibt zwei Lösungen für diese Art von Problem. Zum einen kann man die Bereichsüberprüfung ausschalten. Das kann aber dazu führen, daß kleinere Fehler nicht entdeckt werden, vor allem, wenn der Code tatsächlich Werte verändert. Eine bessere Lösung stellt die kurzgeschlossene Boolesche Auswertung dar, die entweder mit *Options/Compiler/Short circuit* oder der Anweisung {$B-} aktiviert werden kann. Auf diese Weise wird der Ausdruck

```
Liste[I] <> Anzahl
```

niemals ausgewertet, wenn I größer Anzahl ist.

Typische Assemblerfehler

Hier stellen wir nun einige der häufigsten Fallstricke der Assemblerprogrammierung vor, ein Thema, das für diejenigen die mit Turbo Assembler arbeiten ebenso interessant ist wie für die, die den in Turbo Pascal integrierten Assembler benutzen. Im letzten Fall sollten Sie das Programmierhandbuch von Turbo Pascal für weitere Informationen zu dem integrierten Assembler nachschlagen.

Rückkehr zu DOS vergessen

In Pascal wird ein Programm automatisch beendet, wenn kein weiterer Code auszuführen ist, auch wenn kein spezieller Befehl zum Beenden des Programms vorliegt. In Assembler gilt das nicht, da nur diejenigen Aktionen ausgeführt werden, die ausdrücklich angegeben wurden. Wenn Sie ein Programm starten, das keinen Befehl für die Rückkehr zu DOS enthält, geht die Ausführung auch dann noch weiter, wenn der Programmcode schon zu Ende ist. Es werden einfach die Speicherstellen ausgeführt, die zufälligerweise hinter dem Programm liegen.

Rücksprung vergessen

Wie Sie wissen, besteht ein korrekter Unterprogrammaufruf aus einem CALL-Befehl, der Durchführung des Unterprogramms und einem Rücksprung aus diesem Unterprogramm. Vergessen Sie nicht den RET-Befehl am Ende der Funktion, damit der Rücksprung zum aufrufenden Programm vorgenommen werden kann. Wenn man ein Programm eingibt, kann es sehr leicht passieren, daß man den RET-Befehl vergißt.

Die Erzeugung des falschen Rücksprungs

Die PROC-Anweisung bewirkt zwei Dinge: Zum einen wird eine Bezeichnung definiert, mit der man die Funktion aufrufen kann, zum anderen wird in dieser Anweisung festgelegt, ob es sich um eine NEAR- oder FAR-Prozedur handelt.

Die RET-Befehle in einer Prozedur sollten schließlich auch mit dem Datentyp der Prozedur übereinstimmen, oder nicht?

Ja und nein. Das Problem liegt darin, daß es möglich und oft auch wünschenswert ist, mehrere Unterprogramme in ein und derselben Prozedur zusammenzufassen. Da diese Unterprogramme keine dazugehörige PROC-Anweisung besitzen, erhalten die RET-Befehle den Datentyp der gesamten Prozedur, der nicht unbedingt der richtige Datentyp für die einzelnen Unterprogramme ist.

Vertauschte Operanden

Für viele Leute scheint die Reihenfolge der Operanden beim 8086 umgekehrt zu sein, und für diese Ansicht gibt es gewiß eine gute Rechtfertigung. Wenn die Zeile

```
mov     ax,bx
```

heißen würde, »Kopiere AX nach BX«, dann könnte man die Zeile schön von links nach rechts durchgehen, und so haben viele Hersteller von Mikroprozessoren ihre Assemblersprachen entworfen. Intel ist die 8086-Assemblersprache jedoch etwas anders angegangen. Für uns heißt die Zeile »Kopiere BX nach AX«, und das ist manchmal etwas verwirrend.

Stack vergessen oder Stack zu klein

In den meisten Fällen wandern Sie über dünnes Eis, wenn Sie für Ihr Programm keinen Stack reservieren. Programme, in denen kein Stack reserviert wurde, laufen manchmal sogar, da der vorher eingestellte Stack vielleicht auf einen nicht verwendeten Speicherbereich zeigt. Aber es gibt keine Garantie, daß diese Programme in jedem Fall laufen, da nicht sichergestellt ist, ob auch nur ein einziges Byte auf dem Stack frei ist. In den meisten Programmen sollte die Anweisung .STACK eingefügt werden, um genügend Speicherplatz auf dem Stack zu reservieren. In Turbo Pascal für Windows können Sie die Stackgröße über *Options/Compiler* verändern. Reservieren Sie in einem Programm soviel Stackspeicher, daß alle Anforderungen an den Stack befriedigt werden können.

Unterprogramme, die verwendete Register löschen

Beim Schreiben eines Assemblerprogramms kann man sich Register sehr gut als lokale Variablen vorstellen, die nur in der gerade bearbeiteten Prozedur verwendet werden. Das geht stellenweise so weit, daß man annimmt, die Register seien beim Aufruf anderer Prozeduren nicht verändert worden. Dem ist aber nicht so, denn Register sind im Grunde globale Variablen, und jede Prozedur kann einige oder alle Register retten oder zerstören.

Falsche Verwendung von bedingten Sprungbefehlen

Die Fülle an bedingten Sprungbefehlen in Assembler (`JE, JNE, JC, JNC, JA, JB, JG` usw.) führt zu einer enormen Flexibilität beim Schreiben von Program-

men, macht es einem aber auch sehr leicht, gegebenenfalls einen falschen Sprungbefehl zu erwischen. Da man in Assembler zudem noch mindestens zwei Befehle benötigt, nämlich einen für den Vergleich und einen für die bedingte Verzweigung, ist die Behandlung von bedingten Anweisungen weitaus weniger gut nachzuvollziehen und viel fehlerträchtiger als in Pascal.

Zeiger und Zähler gehen einen Schritt zu weit

Stringbefehle haben eine seltsame Eigenschaft. Nach ihrer Ausführung werden die Zeiger um eine Adresse weitergeschoben. Das kann man in den wiederholten Stringbefehlen leicht übersehen, vor allem in `REP SCAS` und `REP CMPS`.

0 ist nicht gleich 0 in CX

Jeder Stringbefehl, den man mit einem Wiederholungsfaktor von 0 in CX verwendet, wird überhaupt nicht ausgeführt. Das kann sehr praktisch sein, weil man das CX-Register vor der Ausführung der Stringoperation nicht extra auf 0 hin überprüfen muß. Leider gibt es aus diesem Grund keine Möglichkeit mehr, in einem Segment mit Byte-Werten auf alle Werte mit einem Befehl zuzugreifen.

Direction-Flag falsch

Wenn ein Stringbefehl ausgeführt wird, werden die damit verbundenen Zeiger – SI, DI oder beide – inkrementiert oder dekrementiert. Das hängt nur vom Status des Direction-Flags ab.

Das Direction-Flag kann mit dem Befehl CLD gelöscht werden, so daß die Zeiger in den Stringbefehlen inkrementiert werden. Mit dem Befehl STD kann man das Direction-Flag setzen, so daß die Zeiger in den Stringbefehlen dekrementiert werden. Wenn das Direction-Flag einmal gesetzt oder gelöscht wurde, bleibt das Flag so lange in der Stellung, bis ein anderer CLD- oder STD-Befehl ausgeführt wird oder die Flags mit POPF oder IRET vom Stack geholt werden. Obwohl es sehr bequem ist, daß das Direction-Flag nur einmal für eine Reihe von Stringbefehlen gesetzt werden muß, kann es durch eine Schlamperei mit dem Direction-Flag zu Fehlern kommen, die völlig unvermittelt auftreten und eventuell sehr schwer zu finden sind, da sie vom Code abhängen, der vorher ausgeführt wird.

Falscher Vergleich in Stringfunktionen

Der Befehl CMPS vergleicht zwei Speicherbereiche, während SCAS den Inhalt des Akkumulators mit einem Speicherbereich vergleicht. Wenn man die Befehle mit REPE modifiziert, dann führen beide Befehle die Vergleiche so lange durch, bis entweder CX 0 wird oder die Operanden eines Vergleichs nicht identisch sind. Leider kann man die beiden Befehle und die Bedeutung der Vorgaben sehr leicht verwechseln.

Standard-Segmente vergessen

Jeder Stringbefehl arbeitet mit bestimmten Standard-Segmenten, wie zum Beispiel dem Quellsegment DS (falls vorhanden) und dem Zielsegment ES (falls vorhanden). Man kann das leicht vergessen und einen STOSB mit dem Daten-

segment durchführen wollen, da dies normalerweise der Platz ist, an dem die Daten mit Befehlen verarbeitet werden, die keine Stringbefehle sind.

Fehlerhafte Umsetzung von Byte- in Word-Operationen

Im allgemeinen ist es empfehlenswert, wenn man die Datentypen in den Stringbefehlen voll ausschöpft (nomalerweise ein Word, beim 80386 ein Doppelword), da die Stringoperationen mit größeren Datentypen oft schneller sind. In der folgenden Umwandlung

```
shr    cx,1
```

eines Byte-Zähler in einen Wort-Zählers geht ein Byte verloren, wenn CX ungerade ist.

Dann dürfen Sie auch nicht vergessen, den Byte-Zähler mit SHR durch 2 zu dividieren. Wenn beispielsweise STOSW mit einem Byte- anstelle eines Word-Zählers arbeitet, können andere Datenbereiche vernichtet und allerhand Probleme verursacht werden.

Die Verwendung mehrerer Segmentvorgaben

Stringbefehle mit mehreren Segmentvorgaben arbeiten nicht zuverlässig und sollten im allgemeinen vermieden werden.

Auf Stringoperanden kann man sich nicht verlassen

In Stringbefehlen kann man optional Operanden angeben, um die Größe der Daten oder andere Segmentregister vorzugeben. Es ist jedoch nicht garantiert, daß man auf die angegebenen Speicherbereiche auch tatsächlich zugreifen kann.

Bei Multiplikationen werden Register zerstört

Eine Multiplikation, ob 8, 16 oder 32 Bit breit, zerstört immer den Inhalt von zumindest einem Register – und natürlich den Teil des Akkumulators, der als Quelloperand diente.

Stringbefehle verändern unerwartet ein Register

Die Stringbefehle (MOVS, STOS, LODS, CMPS und SCAS) können einige Flags und bis zu drei Register mit einem Befehl verändern. Wie beim Befehl MUL sind die Veränderungen aus der Syntax des Befehls nicht ersichtlich. Wenn Sie Stringbefehle verwenden, sollten Sie sich merken, daß SI oder DI oder manchmal sogar beide in Abhängigkeit vom Direction-Flag entweder inkrementiert oder dekrementiert werden. CX wird ebenfalls zumindest einmal und möglicherweise bis zu 0 dekrementiert, wenn der Wiederholungsbefehl REP vor dem Stringbefehl steht.

Falsche Erwartungen beim Carry-Flag

Während manche Befehle unerwartet ein Register oder ein paar der Flags verändern, werden die Flags in anderen Befehlen nicht verändert, wo man es vielleicht eher erwarten würde.

Zu lang gefackelt mit den Flags
Das Statusregister ist nur solange gültig, bis es vom nächsten Befehl verändert wird, was meist nicht allzu lange dauert. Am besten verarbeitet man die Flags also sofort, um mögliche Fehler von vornherein vermeiden.

Verwechslung Speicheroperanden – konstante Operanden
Ein Assemblerprogramm kann sich entweder auf den Offset oder den Inhalt einer Speichervariablen beziehen. Es ist jedoch aufgrund der Assemblersyntax nicht mehr sofort ersichtlich, wie man diese beiden Arten von Zugriffen vornehmen kann, und deshalb werden Offsets und Referenzen auf eine Speichervariable oft verwechselt.

Segmentüberlauf
Einer der schwierigsten Aspekte bei der Programmierung des 8086 ist die Tatsache, daß der Speicher nicht als großes Array an Bytes vorliegt, sondern nur in 64-KByte-Abschnitten relativ zu einem Segmentregister verfügbar ist. Durch die Verwendung von Segmenten kann es zu Fehlern kommen, die nur sehr schwer zu finden sind, denn wenn ein Programm hinter der Grenze eines Segments zu lesen versucht, wird leider wieder von vorne angefangen.

Gruppenvorgaben in Operanden und Tabellen vergessen
Mit Hilfe von Segmentgruppen kann man logisch getrennte Daten auf viele Bereiche verteilen, ohne daß man jedesmal ein Segmentregister neu laden muß, wenn man von einem dieser logischen Bereiche in einen anderen wechseln will.

Vergessen, in einem Interrupt-Handler alles zu retten
Jeder Interrupt-Handler sollte den Inhalt von allen Registern sichern. Obwohl es auch möglich ist, nur die verwendeten Register zu sichern, ist es immer gut, wenn man alle Register zu Anfang des Interrupt-Handlers auf den Stack legt und am Ende wieder alle Register herunterholt. Es kann schließlich vorkommen, daß Sie den Code des Interrupt-Handlers eines Tages umschreiben und einige zusätzliche Register verwenden, dabei aber vergessen, diese Register zu retten.

Richtigkeit überprüfen

Zu überprüfen, ob ein Programm mit zulässigen Eingaben funktioniert, ist nur die halbe Arbeit des Testens. In den folgenden Abschnitten besprechen wir einige wichtige Testsituationen, unter denen man jedes Programm testen sollte, bevor man ihm eine korrekte Funktionsweise bescheinigt.

Grenzfälle und Einschränkungen

Wenn Sie meinen, daß eine Routine mit einem bestimmten Bereich von Eingabewerten funktioniert, dann sollten Sie sie mit Daten testen, die in den Grenzbereichen der zulässigen Eingaben liegen. Wenn Sie zum Beispiel eine Routine haben, die eine Liste von 1 bis 20 Objekten darstellt, dann sollten Sie überprüfen, ob Sie auch korrekt läuft, wenn genau ein Objekt in der Liste ist und exakt 20 Objekte in der Liste sind. Dabei kann sich dann einer der »eins-zuviel«- und »eins-zuwenig«-Fehler ergeben.

Falsche Eingabedaten

Sobald Sie der Ansicht sind, daß die Routine über den gesamten zulässigen Bereich von Eingabewerten funktioniert, sollten Sie feststellen, was die Routine macht, wenn unzulässige Daten eingegeben werden. Sie sollten überprüfen, ob die Routine die Eingabedaten als unzulässig zurückweist, auch wenn die Daten sehr nahe bei den zulässigen Daten liegen. Die Routine, die vorher die Werte von 1 bis 20 akzeptiert hat, sollte beispielsweise die Werte 0 und 21 zurückweisen.

Leere Eingabe

Das ist ein sowohl beim Testen als auch bei der Programmentwicklung häufig übersehener Bereich. Wenn Sie ein Programm schreiben, das eine vernünftige Voreinstellung aufweist, auch wenn Eingabedaten ausgelassen werden, erhöhen Sie die Benutzerfreundlichkeit Ihres Programms enorm.

Die Fehlersuche als Teil der Programmentwicklung

Wenn Sie mit dem Entwurf Ihres Programms anfangen, können Sie die Fehlersuche bereits planen. Einer der wesentlichen Punkte bei der Programmentwicklung betrifft den Grad der Datenüberprüfung in den einzelnen Teilen Ihres Programms.

Wenn Sie viele Überprüfungen vornehmen, erhalten Sie ein unverwüstliches Programm, das Sie über einen Fehler informiert, aber vernünftig weitermacht. Diese Art von Programmen kann sehr leicht zu debuggen sein, da Sie bereits von den Routinen selbst über unzulässige Daten informiert werden, bevor sich die Gefahr fortpflanzt.

Sie können auch ein Programm implementieren, das nur eine geringfügige oder gar keine Überprüfung der Ein-/Ausgabedaten vornimmt. Ihr Programm wird dann schneller und kompakter sein, aber mit falschen Eingabedaten oder einem kleinen Fehler kann man das Programm leicht aussteigen lassen. Diese Programme sind am schwierigsten zu debuggen, da sich ein kleines Problem während der Ausführung möglicherweise erst viel später auswirkt. Dadurch wird es schwierig, Ort und Stelle des ursprünglichen Fehlers zu finden.

Die meisten Programme enden als eine Mischform der beiden Methoden, wobei Eingabedaten von externen Quellen (wie zum Beispiel vom Anwender oder aus einer Datei) mit weitaus mehr Vorsicht behandelt werden als Daten, die intern von einer Routine an eine andere weitergegeben werden.

Beispiel für eine Debuggersitzung

In diesem Beispiel für die Fehlersuche mit einem Debugger setzen wir einige Methoden, die in den Kapiteln zuvor besprochen wurden, in die Praxis um. Das Programm, TDDEMOB.PAS, das wir debuggen werden, ist eine bewußt fehlerhafte Version des Demonstrationsprogramms aus Kapitel 3 (TDDEMO.PAS), mit dem Unterschied, daß bewußt einige Fehler eingebaut wurde. Auch dieses Programm wickelt Ein- und Ausgaben über die Unit *WinCrt* ab.

Die beiden Dateien, TDDEMOB.PAS und TDDEMOB.EXE, für die Demonstration zur Fehlersuche sollten in Ihrem aktuellen Verzeichnis stehen. (Das B in den Dateinamen steht für »buggy« [engl. fehlerhaft].)

Fehler suchen

Bevor wir mit dem Debuggen beginnen, wollen wir das fehlerhafte Pascal-Programm einmal aufrufen, um zu sehen, was falsch läuft. Das Programm ist bereits compiliert und auf den Originaldisketten verfügbar.

Um den Quellcode besser ansehen zu können und den Ablauf und das Debuggen des Programms zu vereinfachen, starten wir Turbo Pascal für Windows. Laden Sie dann über *File/Open* die Datei TDDEMOB.PAS.

Wenn die Quelldatei im Editorfenster geladen ist, starten Sie das Programm, entweder über *Run/Run* oder das Tastenkürzel [Ctrl][F9].

Sie sehen, daß sich ein Fenster öffnet, dem der Programmname als Titel zugeordnet wurde, und in dem ein Cursor zur Eingabe auffordert. Vergrößern Sie dieses Fenster zunächst auf seine maximale Größe, damit genügend Platz für die Ausgaben des Programms vorhanden ist. Geben Sie dann den folgenden Text genau so ein:

```
ABC DEF GHI
abc def ghi
```

Mit einer Leerzeile beenden Sie Ihre Eingabe. TDDEMOB gibt dann folgende Analyse Ihrer Eingabewerte aus:

```
9 char(s) in 3 word(s) in 2 line(s)
Average of 0.67 Words per line

Word length:   1   2   3   4   5   6   7   8   9  10
Frequency:     0   0   3   0   0   0   0   0   0   0
Letter:        M
Frequency:     1   1   1   1   1   1   1   1   1   0   0   0   0
Word starts:   1   0   0   1   0   0   1   0   0   0   0   0   0

Letter:        Z
Frequency:     0   0   0   0   0   0   0   0   0   0   0   0   0
Word starts:   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Wir haben es bei dieser Ausgabe mit vier verschiedenen Fehlern zu tun:

1. Die Anzahl der Wörter ist falsch (3 statt 6)

2. Die Anzahl der Wörter pro Zeile ist falsch (0.67 statt 3.0)

3. Die Spaltenüberschrift der zweiten und dritten Tabelle zeigt nur jeweils einen Buchstaben an (statt A..M und N..Z).

4. Sie haben zwei Zeilen eingegeben, von denen jede einen Buchstaben von A..I enthält, aber die Häufigkeitstabelle für die Buchstaben zählt jeweils nur einen einzigen Buchstaben.

An dieser Stelle können Sie das Fenster schließen und zu Turbo Pascal zurückkehren.

Strategie festlegen

Ihre erste Aufgabe besteht darin, festzulegen, welches Problem Sie zuerst angehen wollen. Eine gute Faustregel lautet, das Problem als erstes zu behandeln, das als erstes auftritt. In diesem Programm wird, nachdem die Daten in der Prozedur Init initialisiert wurden, die Tastatureingabe mit der Funktion GetLine gelesen. Die Eingabe wird dann von ProcessLine weiterverarbeitet, bis der Benutzer eine Leerzeile eingibt. ProcessLine geht jeden String durch und paßt die globalen Zähler an. Anschließend werden die Ergebnisse durch die Prozedur ShowResults dargestellt.

Die durchschnittliche Anzahl der Wörter pro Zeile wird von ShowResults berechnet, wobei die Anzahl der Zeilen und Wörter verwendet wird. Da die Anzahl der Wörter auch falsch zu sein scheint, sollten Sie am besten in ProcessLine nachsehen, wie NumWords berechnet wird. Obwohl NumWords falsch ist, gibt das Ergebnis von 0,67 Wörtern pro Zeile keinen Sinn. Es ist wahrscheinlich ein Fehler in der Berechnung von ShowResults, die wir uns anschauen müssen.

Die Spaltenüberschriften für alle Tabellen werden über ShowResults ausgegeben. Sie sollten warten, bis die Hauptschleife beendet ist, bevor Sie den zweiten oder dritten Fehler suchen. Da die Anzahl der Buchstaben und Wörter falsch ist, kann man darauf wetten, daß in ProcessLine etwas falsch ist, und dort sollten Sie den ersten und vierten Fehler suchen.

Nachdem Sie über das Problem nachgedacht und eine Strategie entwickelt haben, ist der Zeitpunkt gekommen, mit dem Debuggen anzufangen.

Der Aufruf von Turbo Debugger

Um den Debugger zu starten, vergewissern Sie sich, daß das Editorfenster, das den Quelltext von TDDEMOB enthält, aktiv ist, und wählen dann *Run/Dubugger*, wodurch TDW aufgerufen wird.

Turbo Debugger lädt das fehlerhafte Programm und stellt dann den Startbildschirm, die Menüs usw. dar. Wenn Sie die Beispielsitzung beenden wollen, können Sie jederzeit [Alt][X] eingeben, um zu Turbo Pascal zurückzukehren. Wenn Sie sich hoffnungslos verstricken, können Sie das Demonstrationsprogramm in jeder Situation mit [Ctrl][F2] erneut laden und wieder von vorn anfangen. (Beachten Sie bitte, daß damit keine Breakpoints oder *Watches*-Variablen gelöscht werden.)

Es gibt zwei Methoden, um an eine Routine wie ProcessLine heranzugehen: Sie können entweder die Routine Schritt für Schritt ausführen und prüfen, ob alles in Ordnung ist, oder Sie halten das Programm an, nachdem ProcessLine alles erledigt hat und schauen dann nach, ob alles mit rechten Dingen zuging. Da sowohl die Anzahl der Buchstaben als auch der Wörter falsch war, müssen Sie ProcessLine wahrscheinlich sorgfältig überprüfen und zusehen, wie die einzelnen Zeichen verarbeitet werden.

Wandern durchs Programm

Sie lassen das Programm also jetzt laufen und arbeiten den Aufruf von ProcessLine schrittweise ab. Es gibt mehrere Möglichkeiten, wie man das machen kann. Sie können viermal F8 drücken (um über die Prozedur und Funktionsaufrufe zu springen) und dann einmal F7 drücken (um den Aufruf von ProcessLine nachzuverfolgen). Sie können den Cursor auch auf die Zeile 231 setzen, F4 drücken (der Befehl *Go to cursor*) und dann F7 drücken, um in ProcessLine einzutreten.

Ob Sie's glauben oder nicht, es gibt sogar noch weit mehr Möglichkeiten in ProcessLine zu kommen, aber versuchen Sie jetzt die folgende: Drücken Sie [Alt][F9]. Daraufhin erscheint ein Dialogfenster, in der Sie die Adresse, zu der Sie gehen wollen, angeben können. Geben Sie nun ProcessLine ein und drücken Sie ↵. Das Programm läuft nun bis ProcessLine. Geben Sie dieselben Daten ein, die Sie auch vorher eingegeben haben (also ABC DEF GHI).

Es gibt hier mehrere Schleifen. In einer äußeren Schleife wird ein ganzer String durchsucht. Innerhalb dieser Schleife gibt es eine andere Schleife, mit der alle Zeichen übersprungen werden, die keine Buchstaben sind, und eine zweite Schleife, um die Wörter und Buchstaben zu verarbeiten. Bewegen Sie den Cursor auf die *while*-Schleife in Zeile 159 und drücken Sie F4 (*Go to cursor*).

Diese Schleife wird solange durchlaufen, bis das Ende des String erreicht ist oder ein Buchstabe gefunden wurde. Die zweite Bedingung wird mit einem Aufruf einer BOOLEAN-Funktion, Buchstabe, überprüft. Drücken Sie F7, um in Buchstabe zu tracen. Buchstabe ist eine verschachtelte Funktion, die einen Zeichenwert entgegennimmt und den Wert *True* zurückgibt, wenn es ein Buchstabe ist, ansonsten wird der Wert *False* zurückgegeben. Ein oberflächlicher Blick genügt bereits, um zu sehen, daß die Funktion nur Großbuchstaben überprüft. Sie sollte aber sowohl die Zeichen von A bis Z als auch von a bis z überprüfen oder die Zeichen zumindest in Großbuchstaben umwandeln, bevor der Vergleich durchgeführt wird.

Ein Blick auf die beiden Eingabezeilen, die ursprünglich eingegeben wurden, liefert einen weiteren Hinweis auf die Fehlerquelle (hinterher ist man nämlich immer schlauer). Sie haben sowohl Groß- als auch Kleinbuchstaben von A bis I eingegeben, aber nur die Hälfte der eingegebenen Buchstaben wurde in den Summen mitgezählt. Nun sehen Sie, warum.

Springen Sie jetzt hinter die Zeile, von der aus Buchstabe aufgerufen wurde: Drücken Sie [Alt][F8]. Damit beenden Sie die aktuelle Funktion. Da die zweite Eingabezeile, abc def ghi, nur Kleinbuchstaben enthielt, wurde jedes Zeichen als Leerzeichen gezählt und nicht beachtet. Deshalb wird sowohl die Anzahl der Buchstaben als auch die Anzahl der Wörter falsch und damit ist das Geheimnis der Fehler #1 und #4 gelüftet.

Das Dialogfenster Evaluate/Modify

Es gibt übrigens eine sehr leistungsfähige Methode, um nachzuprüfen, ob Buchstabe fehlerhaft ist. Rufen Sie das Dialogfenster *Evaluate/Modify* zum Auswerten eines Ausdrucks mit [Alt][D][E] auf und geben Sie folgenden Ausdruck ein:

```
Buchstabe ('a') = Buchstabe ('A')
```

Es sind beides Buchstaben, aber die Funktion liefert als Ergebnis FALSE, womit bewiesen wäre, daß Sie nicht wie gewünscht funktioniert. (Im Dialogfenster *Evaluate/Modify* und im *Watches*-Fenster können Sie Ausdrücke auswerten, Zuweisungen durchführen oder, wie hier praktiziert, Prozeduren und Funktionen aufrufen. Weitere Informationen finden Sie in Kapitel 6 »Programmdaten untersuchen und ändern«.)

Untersuchungen

Zwei Fehler sind gefunden, es bleiben aber noch drei. Der Fehler #2 ist viel leichter zu finden, als die beiden vorigen. Drücken Sie [Alt][F8], um ProcessLine zu verlassen, bewegen Sie den Cursor dann in Zeile 207, und drücken Sie F4, um das Programm bis zur Cursorposition laufen zu lassen.

TDDEMOB wird Sie nach einem String fragen. Geben Sie abc def ghi ein und drücken Sie einmal ↵. Drücken Sie noch einmal ↵, wenn das Dialogfenster erscheint. Nun geben Sie F7 ein, um zu ShowResults zu gelangen.

Vergessen Sie nicht, daß Sie herausfinden müssen, warum die durchschnittliche Anzahl der Wörter pro Zeile falsch ist. Die erste Zeile in ShowResults berechnet die Anzahl der Zeilen pro Wörter statt der Wörter pro Zeile. Es ist klar, daß dieser Ausdruck umgedreht werden muß.

Da Sie schon mal hier sind, sollten Sie unbedingt überprüfen, ob NumWords und AnzahlZeilen die erwarteten Werte besitzen. AnzahlZeilen sollte den Wert 2 haben und – da Sie den Fehler in Buchstabe zwar gefunden, aber noch nicht verbessert haben – NumWords sollte den Wert 3 haben. Gehen Sie mit dem Cursor auf AnzahlZeilen und drücken Sie [Alt][F10][I], um die Variable zu untersuchen. Das *Inspector*-Fenster zeigt Ihnen die Adresse, den Datentyp und den aktuellen Wert von AnzahlZeilen sowohl dezimal als auch hexadezimal. Der Wert beträgt tatsächlich 2, so daß Sie weitermachen und zu NumWords schauen können. Drücken Sie Esc, um das *Inspector*-Fenster zu schließen, und gehen Sie dann mit dem Cursor zu NumWords und drücken Sie [Alt][F10][I] noch einmal ([Ctrl][I] hätte den gleichen Effekt). Zum Glück hat auch NumWords den erwarteten (falschen) Wert von 3, und Sie können weitermachen.

Können Sie wirklich weitermachen? Es gibt nämlich noch ein anderes Problem mit dieser Berechnung, und das ist gar nicht auf Ihrer Liste. Es wird nämlich nicht überprüft, ob der zweite Term ungleich 0 ist, bevor die Division ausgeführt wird. Wenn Sie das Programm ganz von Anfang an starten und überhaupt keine Daten eingeben (indem Sie nur ↵ drücken), wird das Programm abstürzen (auch nachdem Sie Divisor und Dividend vertauscht haben).

Um das nachzuprüfen, drücken Sie Esc. Schließen Sie das *Inspector*-Fenster mit [Alt][R][P] und beenden Sie die aktuelle Debuggersitzung. Nun drücken Sie F9, um das Programm von Anfang an zu starten und geben bei der Eingabeaufforderung von TDDEMOB nur ↵ ein. Das Programm wird beendet, und Sie erhalten eine Fehlermeldung mit dem Laufzeitfehler. Sie sollten die Anweisung folgendermaßen verändern:

```
if NumLines <> 0 then
  AvgWords := NumWords / NumLines
else AvgWords := 0;
```

Soviel zu den Fehlern #2 und #2a. Wenn Sie schon ein *Inspector*-Fenster geöffnet haben, sollten Sie es auch einmal verwenden, um eine Datenstruktur durchzugehen. Bewegen Sie den Cursor zur Deklaration von LetterTable in Zeile 50. Stellen Sie den Cursor auf das Wort LetterTable, und drücken Sie [Alt][F10][I]. Sie sehen nun einen Record, der 26 Elemente umfaßt. Verwenden Sie die Cursortasten, um durch jedes Element des Arrays zu rollen, und drücken Sie die

Taste ⏎, um in eines der Array-Elemente hineinzugehen. Das ist eine sehr praktische Methode, um Ihre Datenstrukturen zu untersuchen.

Beobachtungen

Sie müssen aber trotzdem noch den kleinen Fehler (#3) in ShowResults ausbügeln. Da Sie das Programm bereits beendet haben, als Sie den Fehler mit der Division durch Null gefunden hatten, müssen Sie sich für einen neuen Anlauf vorbereiten. Drücken Sie [Alt][R][P] (*Programm reset*), um das Programm neu zu laden. Anschließend geben Sie [Alt][F9], ShowResults und ⏎ ein. Nun können Sie die allzu bekannte Zeile ABC DEF GHI eingeben und noch einmal drücken ⏎. Schließlich geben Sie abc def ghi ein und drücken zweimal ⏎. Turbo Debugger sollte nun in ShowResults anhalten.

ShowResults verwendet eine verschachtelte Prozedur, ShowLetternfo, um die Buchstabentabellen auszugeben. Bewegen Sie den Cursor auf Zeile 103, drücken Sie F4 und dann F7, um in ShowLetternfo einzusteigen.

Es gibt hier drei **for**-Schleifen. Die erste gibt die Spaltenüberschriften aus, die zweite und die dritte sind für die Häufigkeitsverteilung zuständig. Verwenden Sie F7, um zur ersten Schleife in Zeile 63 zu gehen. Positionieren Sie den Cursor über FromLet und ToLet und betätigen Sie [Alt][F10][I], um deren Werte zu überprüfen. Diese Werte machen einen guten Eindruck (in der ersten Variablen steht ein A und in der zweiten ein M). Schauen Sie sich den Ausgabebildschirm mit [Alt][F5] an. Drücken Sie eine beliebige Taste, um wieder zum *Module*-Fenster zurückzukehren.

Wenn Sie durch eine Schleife wie diese wandern, ist das *Watches*-Fenster sehr praktisch. Positionieren Sie den Cursor über ch und drücken Sie [Ctrl][W]. Verwenden Sie nun F7, um die for-Schleife schrittweise abzuarbeiten. Wie erwartet, kommen Sie zur Write-Anweisung in Zeile 71. Wenn Sie in das *Watches*-Fenster sehen, bemerken Sie zu Ihrer Überraschung, daß z dort bereits den Wert M hat. (Es wurde also sofort die gesamte Schleife ausgeführt!) Es steht nämlich hinter dem do ein Strichpunkt zuviel, so daß die for-Schleife dreizehnmal abgearbeitet wird, ohne etwas zu tun. Wenn das Programm schließlich zur Write-Anweisung kommt, steht in z der aktuelle Wert, nämlich M. Wenn man den zusätzlichen Strichpunkt entfernt, ist der Fehler #3 behoben.

Das Ende

Das sind alle (bekannten) Fehler dieses Programms. Vielleicht finden Sie sogar noch mehr, wenn Sie den Code noch intensiver unter die Lupe nehmen. Sie können die Fehler nun ausbessern und das Programm neu compilieren (die Fehler sind freundlicherweise mit zwei Sternchen ** gekennzeichnet) oder TDDEMO.PAS, die fehlerfreie Version dieses Programms aus Kapitel 3, »Einführung in das Debuggen«, verwenden.

16. Ein ObjectWindows-Programm debuggen

Die Beispielprogramme, die wir in diesem Kapitel verwenden, wurden mit den ObjectWindows-Klassen geschrieben, die das Programmieren unter Windows wesentlich erleichtern. Alle Techniken und Vorgehensweisen, die in Kapitel 15 »Debuggen eines Standard-Pascal-Programms« beschrieben wurden, sind auch auf die Windows-Programme in diesem Kapitel anwendbar.

In diesem Kapitel beziehen wir uns auf die Programme TDODEMO und TDODEMOB (das B steht für »*buggy*«, also fehlerhaft). TDODEMOB enthält mehrere Fehler, die wir nacheinander beim Durcharbeiten dieses Kapitel beheben werden.

Bevor Sie fortfahren, sollten Sie einmal TDODEMO unter Windows starten, und sich mit dessen Funktion und Aufbau vertraut machen. Zum Aufruf verwenden Sie entweder im Programm-Manager die Menübefehle *Datei/Ausführen...* oder Sie fügen das Programm einer Programmgruppe hinzu und klicken dann auf das Symbol.

Über das Programm

TDODEMO ist ein ObjectWindows-Programm, mit dem Sie unter Verwendung der Maus in verschiedenen Farben auf dem Bildschirm zeichnen können. Wenn Sie die linke Maustaste gedrückt halten und die Maus bewegen, zeichnet das Programm zum Beispiel eine Linie auf den Bildschirm. Mit einem Klick der rechten Maustaste können Sie den Bildschirminhalt löschen. TDODEMO stellt auch eine Menüleiste bereit, die Ihnen aus den vier Stiftfarben Rot, Grün, Blau und Schwarz zu wählen erlaubt.

Um eine Linie zu zeichnen, drücken Sie die Maustaste, bewegen die Maus entsprechend und lassen die Maustaste wieder los. Diese Aufgabe realisiert das Programm sehr unkompliziert, indem es die ObjectWindows-Bibliothek und *dynamische virtuelle Methoden* verwendet. Eine dynamische virtuelle Methode ist eine virtuelle Methode mit einem an die Methodendeklaration angehängten numerischen Wert.

Da Turbo Pascal für Windows Botschaftsnamen als numerische Konstanten definiert, können Sie den Windows-Botschaftsnamen als Bezeichner einer

dynamischen Methode verwenden. ObjectWindows kann dann die Methode immer aufrufen, wenn das Fenster, für das die Methode deklariert wurde, eine Botschaft erhält, die dem Bezeichner der Methode entspricht. Wenn es keine Methode mit einem Bezeichner gibt, der der Windows-Botschaft entspricht, ruft ObjectWindows die voreingestellte Fenster-Prozedur auf.

Um zum Beispiel ein Methode zu entwickeln, die auf WM_MOUSEMOVE-Botschaften reagiert, können sie in einem Fensterobjekt eine Methode definieren, die folgendermaßen aufgebaut sein kann:

```
procedure WMMouseMove (var Msg: TMessage); virtual WM_MOUSEMOVE;
```

Wie Sie sehen können, folgt der Bezeichner WM_MOUSEMOVE unmittelbar der Anweisung virtual.

Der Typ *TMessage* enthält die Windows-Prozedurparameter *wParam* und *lParam*. Diese Parameter speichern oft ergänzende Informationen zu der Botschaft, zum Beispiel die Position der Maus.

In den folgenden Abschnitten wird erklärt, wie TDODEMOB arbeitet. Die darin enthaltenen Fehler werden absichtlich übersprungen, damit Sie sie später in Ruhe ausfindig machen können. Es ist empfehlenswert, zuerst Turbo Pascal für Windows zu starten und die Datei TDODEMOD.PAS zu öffnen, damit Sie sich die einzelnen Anweisungen parallel zur folgenden Beschreibung ansehen können.

Die Definition des Fenstertyps ScribbleWindow

Der Fenstertyp *ScribbleWindow* ist wie folgt definiert:

```
type
  ScribbleWindow = object(TWindow)
    HandleDC: HDC;
    ButtonDown: Boolean;  { Flag: linke Taste gedrückt }

    constructor Init (aParent: PWindowsObject; aTitle: PChar);
    procedure WMLButtonDown (var Msg: TMessage);
      virtual WML_BUTTONDOWN;
    procedure WMLButtonUp (var Msg: TMessage);
      virtual WML_BUTTONUP;
    procedure WMMouseMove (var Msg: TMessage);
      virtual WM_MOUSEMOVE;
    procedure WMRButtonDown (var Msg: TMessage);
      virtual WMR_BUTTONDOWN;
  end;
```

Der Typ *ScribbleWindow* definiert ein Fensterobjekt, das auf die folgenden Benutzereingaben reagiert:

- Mausbewegung
- Linke Maustaste drücken und loslassen

- Rechte Maustaste drücken

Es gibt zwei Instanzvariablen, *HandleDC* und *ButtonDown*, die die Geräteklasse beziehungsweise den Status der Maustasten angeben.

Der Konstruktor Init
Der Konstruktor *Init* ruft den Standardkonstruktor *TWindow.Init* auf, wodurch sich das Fenster genauso verhalten kann wie ein beliebiges anderes *TWindow*-Objekt und gleichzeitig die Variable *ButtonDown* auf *False* setzt.

Die Methode WMLButtonDown
Wenn der Benutzer die Maustaste im *Scribble*-Fenster drückt und anfängt zu zeichnen, erhält das Fenster eine WML_BUTTONDOWN-Botschaft, die ObjectWindows dazu veranlaßt, die Methode *WMLButtonDown* aufzurufen (da dieser Methode als Indexwert WM_LBUTTONDOWN zugeordnet wurde). *WMLButtonDown* bewegt den Stift zur aktuellen Position der Maus und speichert in der Variablen *ButtonDown* einen Wert, der angibt, daß die Maustaste gedrückt ist. Die Prozedur sollte weitere Windows-Funktionen aufrufen, worauf wir später zurückkommen werden.

Die Methode WMMouseMove
Sobald der Benutzer beginnt, die Maus im Fenster zu bewegen, werden auch die ersten WM_MOUSEMOVE-Botschaften an das Fenster geschickt. Dadurch wird ObjectWindows veranlaßt, die Methode *WMMouseMove* aufzurufen. Wenn der Benutzer die linke Maustaste niedergedrückt hält, zeichnet das Programm jedesmal eine Linie wenn die Maus bewegt wird. Wenn der Benutzer keine Maustaste gedrückt hat, passiert nichts.

Die Methode WMButtonUp
Beendet der Benutzer seine Zeichnung und läßt die Maustaste los, erhält das Fenster die Botschaft WM_LBUTTONUP, was wiederum dazu führt, daß ObjectWindows die Methode *WMLButtonUp* aufruft. Das Programm setzt die Variable *ButtonDown* auf *False* und gibt die Geräteklasse frei, die mit dem Fenster verbunden ist.

Die Methode WMButtonDown
Wenn der Benutzer die rechte Maustaste drückt, um den Bildschirminhalt zu löschen, ruft ObjectWindows die Methode *WMRButtonDown* auf, die wiederum die Windows-Prozedur *UpdateWindow* ruft. Der Aufruf dieser Prozedur löscht den Inhalt des Fensters.

Hinzufügen von Farben mit CScribbleWindow

Das Programm definiert nun ein Objekt, das dem Benutzer erlaubt zu zeichnen. Um Farben zu verwenden, wäre es zum Beispiel möglich, dem Typ *ScribbleWindow* weitere Methoden hinzuzufügen. Es ist aber wahrscheinlich

besser, stattdessen die Möglichkeit der Vererbung einzusetzen und ein neues Objekt aus *ScribbleWindow* abzuleiten. Das neue Objekt nennen wir *CScribbleWindow* und es wird folgendermaßen deklariert:

```
CScribbleWindow = object(ScribbleWindow)
  thePen: HPen;
  constructor Init (aParent: PWindowsObject; aTitle: PChar);
  destructor Done; virtual;
  procedure SelectRedPen (var Msg: TMessage);
    virtual cm_First + RedMenu;
  procedure SelectGreendPen (var Msg: TMessage);
    virtual cm_First + GreenMenu;
  procedure SelectBluePen (var Msg: TMessage);
    virtual cm_First + BlueMenu;
  procedure SelectBlackPen (var Msg: TMessage);
    virtual cm_First + BlackMenu;
  procedure SelectRedPen (var Msg: TMessage);
    virtual cm_First + RedMenu;
  procedure WMLButtonDown (var Msg: TMessage);
    virtual WMLButtonDown;
  procedure GetWindowClass (var AWndClass: TWndClass); virtual;
  function GetClassName: PChar; virtual;
end;
```

Das Feld thePen

Dieses neue Objekt besitzt die Instanzvariable *thePen*, die die aktuelle Stiftfarbe speichert. Unter Verwendung dieser Variablen stellen die *CScribbleWindow*-Methoden die Stiftfarbe ein.

Der Konstruktor Init

Der Konstruktor *Init* ruft zuerst den Konstruktor seines Vorfahren auf und weist dann dem Feld *Attr.Menu* des Fensters das Handle zu, das durch den Aufruf der Funktion *LoadMenu* zurückgeliefert wird. Damit besitzt dieses Fenster ein Menü, das die Funktion *LoadMenu* aus der Ressourcendatei lädt. Schließlich initialisiert der Konstruktor einen Stift mit der Farbe Schwarz.

Der Destruktor Done

Der Konstruktor *Done* ruft den Destruktor seines Vorfahren auf und gibt den vorher zugeordneten Stift wieder frei.

Die Stiftfarben-Methoden

Es gibt vier Methoden, die die Stiftfarben durch Löschen des aktuellen Stifts und Erzeugen eines neuen Stifts mit der gewünschten Farbe einstellen. Die Methoden sind funktionell gleich und unterscheiden sich nur durch die gewählten Farben.

Die Methode WMLButtonDown

Jetzt muß nur noch der gewählte Stift dem Gerätekontext zugeordnet werden, bevor mit dem Zeichnen begonnen werden kann. Da das eigentliche Zeichnen nach dem Drücken der linken Maustaste beginnt, ist es notwendig, die dynamische virtuelle Methode *WMLButtonDown* neu zu deklarieren, damit diese Methode und nicht etwa die des Vorfahren *ScribbleWindow* aufgerufen wird.

Die neue Version von *WMLButtonDown* ruft zuerst die *WMLButtonDown*-Methode des Vorfahren auf und ordnet dann *thePen* dem aktuellen Gerätekontext zu.

Erstellen des Programms

Um ein Programm zu erstellen, das das Fenster *ScribbleWindow* verwendet, ist es notwendig ein Objekt basierend auf dem in *ObjectWindows* definierten Objekt *TApplication* zu erstellen. Der Zweck dieses Objekts *CScribbleApplication* ist es, die Methode *InitMainWindow* erneut zu deklarieren, so daß das Programm ein Hauptfenster mit den Eigenschaften von *CScribbleWindow* erzeugen kann.

Sie haben jetzt also einen Überblick, wie das Programm arbeitet und können mit dem eigentlichen Debuggen beginnen. Viel Spaß!

Debuggen des Programms

Falls Sie dies nicht schon bereits getan haben, starten Sie nun Turbo Pascal für Windows, laden TDODEMOB.PAS, compilieren das Programm und lassen es ablaufen. Wenn Sie die Maustaste drücken, um etwas zu zeichnen, stellen Sie fest, daß nichts passiert – der erste Fehler!

Suchen des ersten Fehlers

Da Linien nur mit *WMMouseMove* gezeichnet werden, ist das vermutlich der Bereich, in dem wir nach diesem ersten Fehler suchen sollten. Es gibt drei mögliche Fehlerbedingungen:

- *WMMouseMove* wird nicht aufgerufen, wenn die Maus bewegt wird.
- Probleme beim Aufruf der Windows-Funktion *LineTo*.
- Die Variable *ButtonDown* wird nicht auf *True* gesetzt.

Laden Sie das Programm in TDW. Dies ist äußerst unkompliziert, wenn Sie TDODEMO.PAS in ein Editorfenster laden (was Sie ja schon getan haben) und dann *Run/Debugger* wählen.

Den Fehler eingrenzen

Zuerst sollten Sie prüfen, ob *WMMouseMove* aufgerufen wird. Setzen Sie dazu den Cursor auf *ScribbleWindow.WMMouseMove* und setzen Sie einen Breakpoint auf die Anweisung begin. Drücken Sie F9, um das Programm ablaufen zu lassen (es ist möglich, daß Sie mehrmals F9 drücken müssen, wenn das Programm zu TDW zurückkehrt).

Sobald das Fenster angezeigt wird, bewegen Sie den Mauszeiger innerhalb des Fensters ohne die linke Maustaste zu drücken. TDW stoppt die Ausführung des Programms und kehrt zu dem in *WMMouseMove* gesetzten Breakpoint zurück (damit das auch funktioniert ist es eventuell notwendig, eine Taste zu drücken, um Windows zu veranlassen, die Botschaft WM_MOUSEMOVE auszugeben). *WMMouseMove* wird aufgerufen. Dies schließt den ersten möglichen Fehlergrund aus. Der Fehler liegt also entweder darin, daß *ButtonDown* nicht den richtigen Wert erhält, oder daß es ein Problem beim Aufruf *LineTo* gibt.

Um diese beiden Möglichkeiten auszutesten, entfernen Sie den Breakpoint von der begin-Anweisung *WMMouseMove*, bewegen den Cursor dann zwei Zeilen nach unten zum Aufruf der *LineTo*-Funktion und setzen dort den neuen Breakpoint. Dann bewegen Sie den Cursor zu *ScribbleWindow.WMLButtonUp* und setzen einen weiteren Breakpoint bei der begin-Anweisung dieser Routine.

Wenn Sie das Programm wieder ablaufen lassen, die Maus mit gedrückter linker Maustaste bewegen und die Ausführung beim Aufruf an *LineTo* unterbrochen wird, ist sicher, daß *ButtonDown* die korrekten Werte erhält und das Problem beim *LineTo*-Aufruf liegt. Denn andernfalls würde das Programm zu *WMLButtonUp* zurückkehren, wenn das Problem bei *ButtonDown* läge.

Setzen Sie das Programm mit F9 fort. Wenn sich der Mauszeiger im Fenster befindet, drücken Sie die linke Maustaste und bewegen die Maus. Da die Ausführung bei *LineTo* unterbrochen wird, ist sicher, daß *ButtonDown* den korrekten Wert erhält. Das Problem muß also beim Aufruf an *LineTo* liegen.

Debuggen von LineTo

Der Methode *LineTo* werden drei Parameter übergeben: Den Bildschirmkontext (*HandleDC*), die x-Position (*Msg.LParamLo*) und die y-Position (*msg.LParamHi*). Windows übermittelt die aktuelle Mausposition bei einer WM_MOUSEMOVE-Botschaft im Feld *lParam* der Variablen *Msg* (diese Variable hat den Typ *TMessage*, ein in ObjectWindows definierter Datentyp). Um gültig zu sein, muß der Bildschirmkontext eine Zahl größer Null sein, der x- und der y-Wert müssen kleiner als die Größe des Fensters sein (in Pixel).

Bewegen Sie den Cursor zum Parameter *HandleDC* von *LineTo* und drücken Sie [Ctrl][I], um seinen Wert zu prüfen. Wie Sie sehen können ist der Wert Null. Damit wird angegeben, daß der Kontext niemals zugewiesen war. Da es nicht möglich ist, in einen Kontext zu schreiben, der nicht zugewiesen ist, haben Sie wahrscheinlich das Problem gefunden. Drücken Sie Esc, um die Prüfung dieses Werts zu beenden.

ScribbleWindow.WMLButtonDown ist für die Einstellung des Fenster zum Zeichnen verantwortlich. Dazu sollte auch die Initialisierung des Bildschirmkontexts für das Fenster gehören. Wenn Sie den Code für WML *ButtonDown* überprüfen, werden Sie sehen, daß *HandleDC* niemals ein Bildschirmkon-

text zugewiesen wurde. Der folgende Ausschnitt zeigt *WMLButtonDown* mit der hinzugefügten Kontext-Initialisierungsanweisung:

```
procedure ScribbleWindow.WMLButtonDown(var Msg: TMessage);
begin
  if not ButtonDown then
  begin
    ButtonDown := True;          { Markiert die Maustaste als gedrückt,}
                                 { so daß eine Linie gezeichnet wird, }
                                 { wenn die Maus bewegt wird. }
    HandleDC := GetDC(HWindow);{ Erzeugt Bildschirmkontext für das }
                                 { Zeichnen}

    MoveTo(HandleDC,             { Bewegt den Zeichnungspunkt }
           Msg.LParamLo,         { zu der Position, an der die }
           Msg.LParamHi);        { Maustaste gedrückt wurde. }
    end;
end;
```

Überprüfung der Korrektur

Entfernen Sie die Breakpoints beim Aufruf an *LineTo* und bei *ScribbleWindow.WMLButtonUp*. Als nächstes lassen Sie das Programm ablaufen und verlassen es und dann auch TDW. (Sollten Sie TDW beenden wollen, ohne daß das zu untersuchende Programm ausgeführt wurde, kann es sein, daß Windows von sich aus dieses Programm startet. In diesem Fall müssen Sie das Programm beenden, bevor Sie wieder zu Turbo Pascal zurückkehren können.) Wenn Sie sich wieder in Turbo Pascal befinden, setzen Sie den Cursor zu *ScribbleWindow.WMLButtonUp* und fügen die Kontext-Initialisierungs-Anweisung ein.

Bevor Sie das Programm compilieren, stellen Sie sicher, daß die Debug-Informationen generiert werden:

1. Wählen Sie *Options/Compiler* und schalten Sie *Debug Information* ein.
2. Wählen Sie *Options/Linker* und schalten Sie *Debug Info in EXE* ein.
3. Wählen Sie *Options/Save*, um Ihre Einstellungen abzuspeichern.

Nun können Sie das Programm erneut compilieren (mit *Compile/Compile* oder [Alt][F9]).

Wenn Sie dann die Maustaste drücken und die Maus bewegen, erscheinen schwarze Linien. Versuchen Sie auch andere Farben über das Menü zu wählen. Rot, Grün und Blau sollte problemlos funktionieren. Wenn Sie aber versuchen, die Farbe wieder auf Schwarz zurückzustellen, wird sich die Farbe des Stifts nicht ändern. Ein zweiter Fehler!

Den Farbzuordnungsfehler lokalisieren

Der wahrscheinlichste Grund für diesen Fehler ist in der *CScribbleWindow*-Methode zu suchen, die den schwarzen Stift setzt: *SelectBlackPen*. Verlassen Sie das Programm, starten Sie wieder TDW und setzen einen Breakpoint bei der begin-Anweisung von *CScribbleWindow.SelectBlackPen*. Lassen Sie dann das Programm laufen und wählen Sie *Pen/Black*. TDW sollte die Ausführung beim Breakpoint unterbrechen. Da dies nicht geschieht, muß der Fehler also woanders liegen.

Es scheint, als ob *SelectBlackPen* niemals aufgerufen wird. Da diese Routine über die Tabelle der dynamischen virtuellen Methoden aufgerufen wird, ist zu vermuten, daß dieser Methode ein falscher Indexwert in die dynamische Methodentabelle zugeordnet wurde.

Setzen eines Fenster-Breakpoints

Wenn der Benutzer einen Menüpunkt wählt, übermittelt Windows eine WM_COMMAND-Botschaft an das Fenster, zu dem das Menü gehört. Der Parameter *wParam* der Meldung enthält den Bezeichner des Menüpunkts, der ausgewählt wurde. Wenn ein ObjectWindows-Fenster eine WM_COMMAND-Botschaft erhält, sucht es den Index der dynamischen Methoden des Fensterobjekts nach dem Wert *cm_First+menu_id* ab. *SelectBlackPen* hat den Index *cm_First+BlackMenu*, wobei *BlackMenu* den Wert 104 hat.

Um herauszufinden, welchen Wert der Parameter *wParam* besitzt, wenn der Benutzer die Option *Pen/Black* wählt, müssen Sie TDW anweisen, die Ausführung zu unterbrechen, wenn eine WM_COMMAND-Botschaft eintrifft. Sie können dann das Programm ablaufen lassen, die Menüauswahl treffen und dann *wParam* überprüfen, um zu sehen, ob ihr Wert dem Wert der Konstanten *BlackMenu* entspricht.

Unterbrechen des Programms mit [Ctrl][Alt][SysRq]

Bevor Sie das Programm anweisen können, beim Eintreffen einer WM_COMMAND-Botschaft zu unterbrechen, müssen Sie das Programm zuerst wieder unterbrechen und die Steuerung an TDW zurückgeben. Sie tun dies, indem Sie [Ctrl][Alt][SysRq] drücken. TDW aktiviert dann das *CPU*-Fenster und gibt an, daß der Kernel-Code von Windows ausgeführt wurde, als Sie das Programm abgebrochen haben.

Da das *Module*-Fenster nicht aktiv ist, müssen Sie die sogenannte Karussell-Methode verwenden, um den Botschafts-Breakpoint zu setzen. Drücken Sie [Alt][F3], um das *CPU*-Fenster zu schließen, und dann [Ctrl][F4], um das *Evaluate/Modify*-Fenster aufzurufen. Geben Sie den Handle-Namen des Fensters (*CSApp.MainWindow^.hWindow*) als den auszuwertenden Ausdruck ein, und drücken Sie dann ↵. In Ergebnis-Fenster erscheint dann ein numerischer Wert, der Wert des Fenster-Handle. Notieren Sie sich diesen Wert und schließen Sie das *Evaluate/Modify*-Fenster mit Esc.

Einstellen des Breakpoints

Als nächstes wählen Sie *View/Windows Messages*. Damit wird das *Windows Messages*-Fenster angezeigt. Der obere linke Ausschnitt zeigt, welches Fenster Sie gerade überprüfen. Der obere rechte Ausschnitt zeigt die Art der Botschaften, die Sie überprüfen wollen. Der untere Ausschnitt enthält die Botschaften, die übermittelt wurden. Im Augenblick sind alle Ausschnitte leer, da noch keine zu überprüfenden Fenster angegeben wurden.

Um ein Fenster zur Überprüfung anzumelden, müssen Sie den Cursor normalerweise in den oberen linken Ausschnitt setzen und dann den Namen des Fenster-Handle eingeben. Da das Menü dem Hauptfenster zugeordnet ist, ist es sinnvoll, die Botschaften, die an dieses Fenster geschickt werden, zu protokollieren (es ist das einzige Fenster in unserem Beispielprogramm, was die Sache vereinfacht). Das Fenster-Handle des Hauptfensters ist *CSApp.MainWindow^.hWindow*.

Wenn jedoch das *Module*-Fenster nicht aktiv ist, müssen Sie stattdessen den Wert des Fenster-Handle, den Sie ja vorher notiert haben, eingeben und dann [Alt][H] drücken. Damit kennzeichnen Sie ihn für TDW als Handle und nicht als Prozedur. Wie Sie sehen, wird der Schalter *Identify by* auf *Handle* gesetzt. Schließlich drücken Sie ⏎ und wählen *OK*, um den Handle-Wert in den Ausschnitt einzufügen.

Hinweis: Sie sollten eine Fenster-Handle erst selektieren, nachdem sie im Programm zugeordnet wurde. Ein Handle wird nach der Initialisierung des Fenster-Objekts zugeordnet, in unserem Fall also nachdem die Methode *CScribbleWindow.SetupWindow* ausgeführt wurde.

Der nächste Schritt besteht darin, festzulegen, daß bei der Botschaft WM_COMMAND unterbrochen werden soll. Setzen Sie den Cursor in den oberen rechten Ausschnitt, und rufen Sie dann mit [Alt][F10] das lokale Menü auf. Wählen Sie *Add*, geben Sie WM_COMMAND ein, drücken Sie [Alt][B], um den Schalter *Break* einzuschalten und drücken Sie dann ⏎. Der obere rechte Ausschnitt zeigt darauf die Meldung *Break on message WM_COMMAND*.

Wenn [Ctrl][Alt][SysRq] nicht funktioniert

[Ctrl][Alt][SysRq] (oder [Strg][Alt][S-Abf] auf manchen Tastaturen) funktioniert nicht auf allen Systemen. Wenn dies der Fall ist, müssen Sie das Programm normal beenden, und dann TDODEMOB mit [Ctrl][F2] erneut laden. Wenn das *Module*-Fenster erscheint, müssen Sie als nächstes einen Breakpoint im Code setzen, so daß das Programm bis zu der Stelle ausgeführt wird, an der das Fenster-Handle zugeordnet wurde, dann wieder zu TDW zurückkehrt, so daß Sie den Botschafts-Breakpoint setzen können.

Um den Breakpoint zu setzen, drücken Sie im *Module*-Fenster [Ctrl][PgUp], um an den Beginn der Datei zu springen. Der Breakpoint wird in der *ScribbleWindow*-Methode *ButtonDown* gesetzt. Drücken Sie [Ctrl][S], um das Dialogfenster *Search* aufzurufen, geben Sie *ScribbleWindow.WMLButtonDown* ein und drücken dann ⏎. Wenn der Cursor auf dem Methodennamen steht, drücken Sie F3, um einen Breakpoint bei der Anweisung begin zu setzen.

Als nächsten Schritt drücken Sie F9, um das Programm ablaufen zu lassen. Wenn das Fenster des Programms erscheint, drücken Sie die linke Maustaste,

um den Programmlauf zu unterbrechen und zu TDW zurückzukehren. Entfernen Sie den Breakpoint mit F2.

Nun ist alles bereit, um den Botschafts-Breakpoint zu setzen. Wie das geht, haben wir schon weiter vorne im Abschnitt »Setzen eines Breakpoints« behandelt. Da das *Module*-Fenster aktiv ist und *CSApp.MainWindow^.hWindow* gewählt ist, können Sie nun den Handle-Namen statt des numerischen Wertes eingeben. Ansonsten ist der Ablauf der gleiche.

Überprüfung von *wParam*

Sie können nun mit der Ausführung des Programms fortfahren, indem Sie F9 drücken.

Wählen Sie aus dem Menü *Pen/Black*. Sobald Sie einen schwarzen Stift gewählt haben, stoppt TDW die Ausführung und aktiviert das *CPU*-Fenster, was anzeigt, daß das Programm Ihren Code nicht ausgeführt hat, als es unterbrochen wurde. Mit F3 schließen Sie das *CPU*-Fenster.

Falls notwendig, können Sie das Fenster *Windows Messages* wieder aufrufen. Vergrößern Sie das Fenster, so daß Sie die vollständige Meldung im unteren Abschnitt sehen können. Sie erkennen, daß das Fenster eine WM_COMMAND-Botschaft mit 204 ($CC) im Parameter *wParam* erhielt. Die Konstante *BlackMenu* hat aber den Wert 104 und nicht 204. Der Grund dafür, daß die virtuelle Methode nicht aufgerufen wurde, war also, daß das Programm nach einem Bezeichner von *cm_First+204* suchte, sein Wert aber tatsächlich *cm_First+104* war. Wenn Sie *BlackMenu* auf 204 ändern, sollte die Auswahl eines schwarzen Stifts problemlos funktionieren.

Wenn Sie die notwendigen Korrekturen vorgenommen haben, sieht die Konstantendeklaration am Beginn des Programms wie folgt aus:

```
const
  PenWidth  = 1;       {Breite der Linie}
  MenuID    = 100;     {ID des Menüs in der Ressourcendatei}
  IconID    = 100;     {ID des Symbols in der Ressourcendatei}
  RedMenu   = 101;     {Wert des Menüs Pen/Red}
  GreenMenu = 102;     {Wert des Menüs Pen/Green}
  BlueMenu  = 103;     {Wert des Menüs Pen/Blue}
  BlackMenu = 204;     {Wert des Menüs Pen/Black}
```

Überprüfung der Korrektur

Lassen Sie das Programm nun ablaufen und verlassen Sie es. Verlassen Sie anschließend auch TDW. Wenn Sie sich wieder in Turbo Pascal befinden, ändern Sie die Konstantendefinition von *BlackMenu*, compilieren das Programm erneut und lassen es noch einmal ablaufen.

Wenn Sie nun in dem Fenster zeichnen, werden Sie wahrscheinlich auf ein weiteres Problem stoßen. Wenn Sie während des Zeichnens die Maus erst aus dem Fenster und dann zurück in das Fenster bewegen, jedoch an einer anderen Stelle, wird eine gerade Linie von dem Punkt, an dem Sie das Fenster verlassen haben, und dem Punkt, an dem Sie wieder ins Fenster eingetreten sind, gezeichnet.

Das Programm sollte eigentlich das Zeichnen beim Verlassen des Fensters unterbrechen und erst wieder damit fortfahren, wenn der Mauszeiger wieder in das Fenster eintritt.

Lokalisieren des Fehlers

Eine Stelle, an der man nach einem solchen Fehler suchen könnte, sind die Windows-Botschaften, die an das Fenster übermittelt werden. Verlassen Sie das Programm und laden Sie TDODEMO.PAS in TDW. Bevor Sie irgendwelche Botschaften auswählen, müssen Sie das Hauptfenster initialisieren lassen, so daß das Fenster-Handle gültig ist. Um das zu tun, setzen Sie einen Breakpoint bei *CScribbleWindow.WMLButtonDown* und drücken dann F9, um das Programm ablaufen zu lassen. Als nächstes drücken Sie die linke Maustaste, um in das *Module*-Fenster von TDW zurückzukehren. Sobald Sie sich im *Module*-Fenster befinden, entfernen Sie den gerade gesetzten Breakpoint.

Protokollieren der Windows-Botschaften

Rufen Sie mit *View/Windows Messages* das Fenster *Windows Messages* auf und tippen Sie *CSapp.MainWindow^.hWindow* ein, um dieses Fenster-Handle in den linken Ausschnitt des Fensters *Windows Messages* einzugeben. Selektieren Sie mit [Alt][H] den Schalter *Handle* und drücken Sie dann ↵.

Als nächstes setzen Sie den Cursor in den oberen rechten Ausschnitt und rufen mit [Ctrl][A] das Dialogfenster *Set Message Filter* auf. Dann setzen Sie einen Breakpoint bei WM_LBUTTONUP, so daß TDW wieder die Programmkontrolle übernimmt, nachdem Sie die das Zeichnen abgeschlossen haben. Besonders interessant für Sie sind die zurückkommenden Botschaften. Die Einstellung von WM_LBUTTONUP hat aber die Einstellung *Log all Messages* gelöscht. Sie müssen also noch einmal das Dialogfenster *Set Message Filter* aufrufen und diesmal die Klasse *Log All Messages* selektieren.

Den Fehler entlarven

Lassen Sie mit F9 TDODEMOB ablaufen. Beginnen Sie mit dem Zeichnen, bewegen Sie dann die Maus aus dem Client-Bereich und an einer anderen Stelle wieder zurück in den Bereich. Um die Anzahl der Meldungen zu reduzieren, bewegen Sie den Cursor aus dem Bereich und gleich wieder in den Bereich. Lassen Sie dann die linke Maustaste los, so daß TDW wieder die Kontrolle übernimmt.

Bevor Sie das Fenster *Windows Messages* betrachten, stellen Sie sicher, daß es die maximal mögliche Größe besitzt (mit F5), damit mehr Botschaften angezeigt werden. Wenn Sie den unteren Ausschnitt des Fensters *Windows Messages* betrachten, bemerken Sie zahlreiche WM_NCHITEST- und WM_SETCURSOR-Botschaften. Dazwischen befinden sich einige WM_LBUTTONDOWN-Botschaft, gefolgt von weiteren WM_MOUSEMOVE-Botschaften und schließlich eine WM_BUTTONUP-Botschaft. Es scheint, daß, wenn sich der Cursor nicht

im Fensterbereich befindet, dieses keine WM_MOUSEMOVE-Botschaften, sondern nur WM_NCMOUSEMOVE-Botschaften erhält.

Es wird immer klarer, wo sich der Fehler versteckt. Das Programm zeichnet von der Position der letzten WM_MOUSEMOVE-Botschaft zur Position der aktuellen WM_MOUSEMOVE-Botschaft. Wenn der Cursor den Fensterbereich verläßt, werden keine WM_MOUSEMOVE-Botschaften mehr übermittelt. Deswegen ist bei der Rückkehr in den Fensterbereich die letzte Position, von der die Linie zur aktuellen Position gezogen wird, die Position, an der der Fensterbereich verlassen wurde.

Den Fehler beheben

Eine mögliche Lösung wäre, festzustellen, wann die Maus den Fensterbereich verläßt, so daß das Programm die letzte Mausposition ignorieren kann und mit dem Zeichnen erst wieder fortfährt, wenn sich der Mauszeiger wieder im Fensterbereich befindet. Dazu wären einige logische Arbeitsschritte notwendig, um festzustellen, wann die Maus den Fensterbereich verläßt und wann sie ihn wieder betritt. Es gibt zum Glück eine einfachere Lösung.

Die Windows-Funktion *SetCapture* erfüllt genau unser Bedürfnis. Diese Funktion weist Windows an, alle mausbezogenen Botschaften an das angegebene Fenster zu senden bis das Programm *ReleaseCapture* aufruft und damit bewirkt, daß das Fenster WM_MOUSEMOVE-Botschaften erhält, anstatt WM_NCMOUSEMOVE-Botschaften, wenn die Maus in den Non-Client-Bereich bewegt wird.

Wenn *SetCapture* in *ScribbleWindow.WMLButtonDown* und ReleaseCapture in *WMLButtonUp* eingefügt wird, zeichnet *WMMouseMove* tatsächlich auch außerhalb des Fensters, wenn die Maus in diesen Bereich bewegt wird. Windows schneidet jedoch die Teile der Zeichnung ab, die sich außerhalb des Fensters befinden.

Diese Veränderungen werden im folgenden Listing zusammengefaßt:

```
procedure ScribbleWindow.WMLButtonDown(var Msg: TMessage);
begin
  if not ButtonDown then
  begin
    ButtonDown := True;                { Markiert die Maustaste }
                                       { als gedrückt, so daß bei }
                                       { einer Mausbewegung }
                                       { eine Linie gezeichnet wird. }
    SetCapture(HWindow);               { Weist Windows an, alle }
                                       { Maus-Botschaften zu dem }
                                       { Fenster zu schicken. }
                                       { Die Methode WMLButtonUp }
                                       { macht dies wieder rückgängig.}
    HandleDC := GetDC(HWindow);        { Erzeugt Bildschirmkontext
                                       { für das Zeichnen }
    MoveTo(HandleDC, Msg.LParamLo,     { Bewegt den Zeichencursor zu }
           Msg.LParamHi);              { der Stelle, an der die Maus- }
```

```
                                    { taste gedrückt wurde. }
    end;
end;

procedure ScribbleWindow.WMLButtonUp(var Msg: TMessage);
begin
  if ButtonDown then
  begin
    ReleaseCapture;                 { Weist Windows an, keine Maus- }
                                    { Botschaften mehr an das }
                                    { Fenster zu schicken. }
    ReleaseDC(HWindow, handleDC);   { Gibt den Bildschirmkontext }
                                    { frei, der mit der Methode }
                                    { WMLButtonDown erzeugt wurde. }
    ButtonDown :=  False;           { Markiert die Maustaste als }
                                    { nicht gedrückt. }
    end;
end;
```

Überprüfung der Korrektur

Lassen Sie das Programm laufen und verlassen Sie es. Verlassen Sie anschließend auch TDW. Wenn Sie sich wieder in Turbo Pascal befinden, geben Sie die Veränderungen an den beiden Routinen ein, compilieren das Programm erneut und lassen es ablaufen. Wenn Sie jetzt im Fenster zeichnen, funktioniert alles. Wenn Sie jedoch versuchen, den Fensterinhalt mit der rechten Maustaste zu löschen, passiert nichts. Schon wieder ein Fehler!

Den Fehler lokalisieren

Verlassen Sie das Programm und laden Sie dann TDODEMOB in TDW. Um das Programm bis zu *WMRButtonDown*, der wahrscheinlichen Fehlerquelle, auszuführen, drücken Sie [Alt][F9] und geben *WMRButtonDown* ein. Zeichnen Sie etwas im Fenster und drücken Sie dann die rechte Maustaste. TDW stoppt den Programmlauf am Beginn von *WMRButtonDown*.

Mit [F7] lassen Sie *WMRButtonDown* schrittweise ausführen und stoppen beim Aufruf von *UpdateWindow*. Der einzige Parameter dieser Methode ist *HWindow*. Sie können davon ausgehen, daß *HWindow* korrekt gesetzt wurde, da es von anderen Methoden erfolgreich verwendet wird. Da wir keinen offensichtlichen Fehler finden, sollten wir einmal verfolgen, ob die WM_PAINT-Botschaft, die von einem Aufruf an *UpdateWindow* geschickt werden sollte, tatsächlich von dem Fenster empfangen wird.

Mittlerweile wissen Sie wahrscheinlich schon ziemlich genau, wie ein Botschafts-Breakpoint auf WM_PAINT gesetzt wird. Falls Sie dennoch Fragen haben, gehen Sie die entsprechenden Erläuterungen weiter oben noch einmal durch.

Nachdem der Botschafts-Breakpoint gesetzt ist, lassen Sie das Programm mit F9 laufen. Da es nicht unterbrochen wird und nicht zu TDW zurückkehrt, steht fest: die Botschaft WM_PAINT wird niemals an das Fenster geschickt. Aus irgendeinem Grund arbeitet *UpDateWindows* nicht wie erwartet.

Analyse der Fehlerquelle

Um diesen Fehler zu beheben ist es notwendig, sich etwas näher mit Art und Weise zu befassen, wie Windows die Funktion *UpdateWindow* handhabt. Wenn ein Programm diese Funktion aufruft, prüft Windows, ob ein Teil des Fensters ungültig ist und neu aufgebaut werden muß. Ist dies der Fall, schickt Windows eine WM_PAINT-Botschaft an das Fenster. Ist dies nicht der Fall, gibt es keinen Grund, Systemressourcen mit unnötigen Botschaften zu belegen, deswegen unternimmt Windows nichts. Aber wie findet Windows heraus, daß das Fenster neu aufgebaut werden muß?

Ein Programm informiert Windows darüber, daß zumindest ein Teil eines Fensters ungültig ist, indem es entweder *InvalidateRect* oder *InvalidateRgn* ruft. Diese beiden Funktionen bestimmen den renovierungsbedürftigen Bereich des Fensters und informieren Windows mit einer WM_PAINT-Botschaft darüber, daß es das Fenster auf den neuen Stand bringen soll. Windows weist der WM_PAINT-Botschaft, die es als Reaktion auf diese Funktionsaufrufe sendet, eine niedrige Priorität zu. Wenn Sie also sicherstellen wollen, daß das Fenster sofort auf den neuesten Stand gebracht wird, sollten Sie die Funktion *UpdateWindow* direkt aufrufen.

Den Fehler beheben

Unser Problem kann gelöst werden, indem ein Aufruf an *InvalidateRect* in *WMRButtonDown* hinzugefügt wird. *InvalidateRect* verwendet drei Parameter: ein Fenster-Handle, das das Fenster identifiziert, einen Zeiger auf ein Rechteck, das den Bereich bezeichnet, der in den Renovierungsbereich einbezogen werden soll, und einen Parameter, der angibt, ob das Rechteck gelöscht werden soll. Sie können für den Zeiger auf das Rechteck nil übergeben. Damit weisen Sie Windows an, daß das gesamte Fenster zum Renovierungsbereich hinzugefügt werden soll. Das folgende Listing zeigt *WMRButtonDown* mit den neuen Anweisungen:

```
procedure ScribbleWindow.WMRButtonDown(var Msg: TMessage);
begin
  InvalidateRect(HWindow, nil, True);
  UpdateWindow(HWindow);
end;
```

Überprüfung der Korrektur

Lassen Sie das Programm ablaufen und verlassen Sie es. Verlassen Sie dann auch TDW. Wenn Sie sich wieder in Turbo Pascal befinden, geben Sie die Änderungen von *WMRButttonDown* ein, compilieren das Programm noch ein-

mal und starten es anschließend. Wenn Sie nun im Fenster zeichnen und die rechte Taste drücken, um die Zeichnung zu löschen, wird dieser Schritt auch tatsächlich durchgeführt.

Damit haben wir endlich alle Fehler gefunden, lokalisiert und behoben. Das Programm läuft nun einwandfrei.

17. Debuggen von Windows-Anwendungen

Das Beispielprogramm, das wir in diesem Kapitel behandeln, verwendet nur die Standardfunktionen, die Windows bereitstellt. Es stützt sich also nicht auf die Objekte der *ObjectWindows*-Bibliothek, die das Programmieren unter Windows wesentlich erleichtern. Das Debuggen einer Windows-Anwendung ist dem Debuggen einer DOS-Anwendung sehr ähnlich. Alle Techniken, die in Kapitel 15 »Ein Standard-Pascal-Programm debuggen« erwähnt werden, können auch für Windows-Programme angewendet werden.

Da eine Windows-Anwendung das Anwendungsprogramm-Interface von Windows (API) benützt, gibt es viele zusätzliche Möglichkeiten für Programmierfehler. In diesem Kapitel soll nicht erklärt werden, wie die Fehlerquellen in der Windows-Programmierung vermieden werden können. Der Zweck dieses Kapitels ist es vielmehr, an einem Anwendungsbeispiel aufzuzeigen, wie die Eigenschaften von TDW zum Auffinden von Fehlern eingesetzt werden.

Die Beispielprogramme

Mit TDW werden drei mausgesteuerte Beispielprogramme mitgeliefert. Diese Programme sind:

- TDWDEMO.PAS, die Quelldatei eines Programms namens *Simple Paint*. Mit diesem Programm können Sie Linien, Ellipsen und Rechtecke in drei Farben und drei verschiedenen Linienstärken zeichnen.
- TDWDEMOA.PAS, ein Programm, das einige Fehler enthält.
- TDWDEMOB.PAS, ein Programm, aus dem der erste Fehler von BCWDEMOA entfernt wurde.

Zusätzlich gibt es noch einige gemeinsame Dateien für die drei Programmversionen:

- TDWDEMO.INC ist die Include-Datei.
- TDWDEMO.RES ist die compilierte Ressourcen-Datei.
- TDWDEMO.ICO enthält das Symbol des Demo-Programms.

Compilieren und Linken der Demoprogramme

Damit Sie den einzelnen Schritten in diesem Kapitel leichter folgen können, stehen Ihnen drei Versionen des Demoprogramms zur Verfügung. Wenn Sie möchten, können Sie die Änderungen an den Programmen auch selbst vornehmen und sie mit Turbo Pascal für Windows compilieren.

Das Programm TDWDEMO starten

Vielleicht möchten Sie, bevor Sie das Debuggen beginnen, ein wenig mit TDWDEMO experimentieren, um eine Vorstellung davon zu bekommen, was das Programm können soll. Rufen Sie Turbo Pascal für Windows auf, indem Sie dessen Symbol zweimal anklicken. Laden Sie dann über *File/Load* die Datei TDWDEMO.PAS in ein Editorfenster. Mit dem Tastenkürzel [Ctrl][F9] erreichen Sie, daß das Programm compiliert und dann sofort gestartet wird.

Debuggen von TDWDEMOA

Das erste Programm, das Sie debuggen, ist TDWDEMOA. Laden Sie die Quelldatei TDWDEMOA.PAS in ein Editorfenster und wählen dann die Menüoption *Run/Debugger*. Dadurch wird TDW aufgerufen. Wenn notwendig compiliert Turbo Pascal das Programm vorher noch, damit der Debugger auf die notwendigen Informationen zugreifen kann.

Wenn der Quelltext im Fenster *Module* von TDW erscheint, drücken Sie F9, um das Beispielprogramm zu starten.

Was Sie sehen, ist der Bildschirm von *Simple Paint*. Der Mauszeiger wird als Stundenglas dargestellt, was anzeigt, daß das Programm gerade arbeitet. Normalerweise würde das Stundenglas schnell verschwinden und durch einen Pfeil ersetzt werden. Da es aber überhaupt nicht verschwindet, muß etwas falsch sein. Zwar können Sie den Mauszeiger auf dem Bildschirm bewegen, aber das ist auch alles.

Um herauszufinden, wo der Fehler liegt, müssen Sie wieder nach TDW zurückkehren. Drücken Sie irgendeine Taste, worauf der Mauszeiger wieder als Pfeil erscheint. Jetzt können Sie das Demoprogramm normal verlassen. Wieder in TDW, drücken Sie [Ctrl][F2], damit das Programm erneut geladen wird.

Was ist zu tun?

Wie finden Sie nun heraus, was mit dem Programm nicht stimmt? Wenn eine Windows-Anwendung beim Startup hängenbleibt oder abbricht, lädt man sie normalerweise wieder ein und arbeitet die Fensterinitialisierungsroutinen schrittweise ab. Wenn Sie bis zur Botschaften-Schleife kommen, können Sie die Windows-Botschaften protokollieren, um zu sehen, welche Botschaften bei dem Programm eingehen.

Aus der Tatsache, daß der Anfangsbildschirm dargestellt wird, können Sie schließen, daß das Programm bis zur Botschaften-Schleife gekommen ist, weil es Windows-Botschaften verarbeiten muß, um den Bildschirm darzustellen. Deshalb können Sie das schrittweise Abarbeiten des Programms auslassen und direkt zur Protokollierung der Botschaften übergehen. Wenn Sie wissen, welche Botschaften bearbeitet wurden, haben Sie zumindest eine Vorstellung davon, welche Teile des Codes verwendet worden sind.

Protokollieren von Botschaften

Diesmal legen Sie vor dem Starten des Programms fest, daß alle Botschaften protokolliert werden sollen. Öffnen Sie das Fenster *Windows Messages* mit dem Befehl *View/Windows messages* und fügen Sie im oberen rechten Ausschnitt *WndProc* hinzu. Im oberen rechten Ausschnitt erscheint »Log all messages« (alle Botschaften protokollieren). Da Sie genau das wollen, brauchen Sie nichts weiter in diesem Fenster tun.

Weil Sie eine unbekannte Zahl von Botschaften protokollieren wollen, kann es vorkommen, daß TDW der für Botschaften verfügbare Platz (200 Botschaften) ausgeht, bevor Sie aus dem Programm zurückkehren. Sie müssen also dafür sorgen, daß diese Botschaften in einer Datei abgelegt werden. Gehen Sie wie folgt vor:

1. Gehen Sie in den unteren Ausschnitt des Fensters *Window Messages* und rufen Sie das lokale Menü auf.

2. Selektieren Sie den Befehl *Send To Log Window*. Wenn er auf *No* gesetzt ist, schalten Sie ihn mit der Eingabetaste ⏎ auf *Yes* um.

3. Gehen Sie mit dem Befehl *View/Log* in das *Log*-Fenster und rufen Sie mit [Alt][F10] das lokale Menü auf.

4. Wählen Sie *Open Log File* und drücken Sie die Eingabetaste ⏎ zur Bestätigung des Standardnamens für die Logdatei, TDWDEMOA.LOG.

Jetzt können Sie das Programm mit [F9] aufrufen. Wenn der Bildschirm aufgebaut ist, drücken Sie [Ctrl][Alt][SysRq], um zu TDW zurückzukehren.

Analysieren des Botschaftenprotokolls

Im unteren Ausschnitt des Fensters *Windows Messages* sehen Sie wahrscheinlich nur eine Menge *WM_PAINT*-Botschaften. Damit Sie alle Botschaften sehen können, die übertragen wurden, rufen Sie mit dem Befehl *View/File* eine Liste der Dateien im aktuellen Verzeichnis auf. Wählen Sie TDWDEMOA.LOG, um die Log-Datei anzuschauen.

Sie sollten eine ziemlich kurze Liste von Fensterinitialisierungs-Botschaften (ca. 16) gefolgt von einer sehr langen Liste von WM-PAINT-Botschaften sehen. Anscheinend hat Windows damit begonnen, die Botschaften für den Aufbau des Anfangsbildschirms zu senden, ist aber bei der Botschaft *WM_PAINT* steckengeblieben. Diese Analyse entspricht dem, was Sie gesehen haben, als das Programm startete: Der Bildschirm wird aufgebaut, aber sonst passiert nichts.

Finden des Fehlers

Was tun Sie also? Sie könnten den Code nach einem Bereich durchsuchen, in dem *WM_PAINT* behandelt wird. Sinnvoller ist es jedoch, an einer geeigneten Stelle einen Botschaftsbreakpoint zu setzen und dann das Programm schrittweise abzuarbeiten, um festzustellen, wo das Problem sitzt.

Das Setzen eines Botschaftsbreakpoints macht auch das Verlassen der Anwendung mit [Ctrl][Alt][SysRq] unnötig, in dessen Folge das schrittweise Abarbeiten eines Programms und ein Ausstieg aus TDW sehr unsicher wären. Da das Programm bei der Botschaft *WM_PAINT* steckenbleibt, stellen Sie TDW so ein, daß bei der Botschaft *WM_PAINT* ein Halt erfolgt und rufen Sie dann das Programm wieder auf. Gehen Sie wie folgt vor:

1. Gehen Sie wieder in den oberen rechten Ausschnitt des Fensters *Windows Messages* und rufen Sie das lokale Menü auf. Wählen Sie *Add*.

2. Das Dialogfenster *Set Message Filter* wird geöffnet. *Single Message* ist bereits gesetzt und der Cursor steht im Textfeld *Single Message Name*. Geben Sie WM_PAINT ein (nur Großbuchstaben, sonst findet TDW nichts), dann wählen Sie *Break* als Aktion.

3. Starten Sie das Programm mit F9.

Das Programm stoppt sofort und Sie befinden sich in der ersten Zeile von *WndProc*. (Sie müssen das Fenster *Windows Messages* löschen, damit Sie den ganzen Code sehen können.) Diese Routine besteht zur Gänze aus einer Switch-Anweisung in Abhängigkeit von den Botschaften, die speziell von dem Programm gehandhabt werden:

```
function WndProc (HWindow : HWnd; Message: Word;
                  wParam: Word; lParam: LongInt): LongInt; export;
begin
```

```
    WndProc := 0;
    case Message of
      wm_Command:
        WndProc := DoWMCommand(wParam);

      wm_LButtonDown:
        DoLButtonDown(HWindow,lParam)

      wm_LButtonUp:
        DoLButtonUp(HWindow,lParam);

      wm_MouseMove:
        DoMouseMove(HWindow,lParam);

      wm_Paint:
        DoPaint(HWindow);
    else
      WndProc := DefWindowProc(HWindow, Message, wParam, lParam);
    end;
  end;
```

Schrittweises Abarbeiten des Programms

Durch Drücken von F7 können Sie das Programm schrittweise abarbeiten. Die Zeilenmarkierung bewegt sich schrittweise bis zu der Anweisung *case WM_PAINT* und verzweigt dann in die Routine *DoPaint*:

```
procedure DoPaint(HWindow: HWnd);
var
  i, SaveROP: Integer;
  HandleDC, hMemDC: HDC;
  theRect, DestRect: TRect;
  theBitmap: HBitMap;
  ps: TPaintStruct;

begin
  if CurrentPoint = 0 then
  begin
    HandleDC := BeginPaint(HWindow, ps);
    { Legt fest, welcher Bereich des Fenster ungültig ist. }
    { Ist kein Bereich als ungültig markiert, wird das     }
    { gesamte Fenster neu ausgegeben.                      }
    GetUpdateRect(HWindow, theRect, False);
    if IsRectEmpty(theRect) then GetClientRect(HWindow, theRect);

    { Erzeugt einen Speicherbereich von der gleichen Größe wie }
    { der ungültige Bereich }
    hMemDC := CreateCompatibleDC(HandleDC);
    theBitmap := CreateCompatibleBitmap(HandleDC,
      theRect.Right - theRect.Left, theRect.Bottom - theRect.Top);
    SelectObject(hMemDC,theBitmap);
```

```
    { Löscht den Bereich }
    BitBlt(hMemDC, 0, 0,
      theRect.Right - theRect.Left, theRect.Bottom - theRect.Top,
      HandleDC, 0, 0, SRCCopy);

    { Zeichnet nur die Formen, die innerhalb des Update-Bereichs liegen }
  for i := 0 to CurrentPoint do
    begin
      IntersectRect(DestRect, thisShape[i].Points, theRect);
      if not IsRectEmpty(destRect) then
        DrawShape(hMemDC,
          thisShape[i].Points.Left - theRect.Left,
          thisShape[i].Points.Top - theRect.Top,
          thisShape[i].Points.Right - theRect.Left,
          thisShape[i].Points.Bottom - theRect.Top,
          thisShape[i].theShape, thisShape[i].PenWidth,
          thisShape[i].PenColor, thisShape[i].Slope);
          { Beachten Sie, daß die Position der gezeichneten   }
          { Formen so verändert wurde, daß sie sich auf die   }
          { obere linke Ecke als Ursprung bezieht. Das ist    }
          { der Punkt (0,0) des Speicherbereichs, der dann    }
          { am Punkt (theRect.Left, theRect.Right) einge-     }
          { blendet wird.                                     }
    end;

    { Jetzt wird der Speicherbereich in den Update-Breich kopiert }
    BitBlt(HandleDC, theRect.Left, theRect.Top,
      theRect.Right - theRect.Left, theRect.Bottom - theRect.Top,
      hMemDC, 0, 0, SRCCopy);

    DeleteDC(hMemDC);
    DeleteObject(theBitmap);
    ReleaseDC(HWindow, HandleDC);
    EndPaint(HWindow, ps);
  end;
end;
```

Wenn Sie mit dem schrittweisen Abarbeiten weitermachen, werden Sie bemerken, daß die einzige Zeile des Codes von *DoPaint*, die ausgeführt wird, die Zeile

```
if CurrentPoint >= 0
```

ist. Kontrollieren Sie die Returns zu der Meldeschleife, wo das Programm die nächste Botschaft, nämlich *WM_PAINT*, aufnimmt und dann wieder die Schleife von *WndProc* und *DoPaint* durchläuft. Offensichtlich macht *DoPaint* etwas falsch, aber was sollte es eigentlich tun?

Analysieren von *DoPaint*

Diese Routine soll, wenn sie zum ersten Mal aufgerufen wird, den ganzen Bildschirm ausmalen oder, wenn etwas auf den Bildschirm gezeichnet wurde, einen Teil des Bildschirms, das aktuelle Rechteck, wieder zeichnen. Um festzustellen, ob etwas gezeichnet wurde oder nicht, prüft *DoPaint* den Wert von *CurrentPoint*, der ursprünglich auf -1 gesetzt war. (*CurrentPoint* zeigt an, wieviele Objekte gezeichnet worden sind). Wenn *CurrentPoint* -1 ist – das ist der Fall, wenn das Programm startet und der Anfangsbildschirm dargestellt wird – braucht die Routine den Inhalt des aktuellen Rechtecks nicht zu übernehmen und neu zu zeichnen. Deshalb wird der Code innerhalb der if-Anweisung übersprungen und Windows veranlaßt, den ganzen Bildschirm neu darzustellen.

Wenn Sie den Wert von *CurrentPoint* mit Hilfe des Fensters *View/Watches* überprüfen, werden Sie feststellen, daß sein Wert -1 bleibt, egal wie oft die Routine durchlaufen wird. Das erscheint logisch, weil Sie nicht die Möglichkeit hatten, etwas zu zeichnen.

Fehlerbeseitigung

An diesem Punkt wird, wenn Sie in *Programming Windows* von Charles Petzold nachlesen, wie Windows die Botschaft *WM_PAINT* behandelt, die Ursache des Problems deutlich. Windows verlangt nach der Botschaft *WM_PAINT* mindestens den Aufruf von *BeginPaint* gefolgt von dem Aufruf von *EndPaint*. Wenn diese Routinen nicht aufgerufen werden, erkennt Windows nicht, daß seine *WM_PAINT*-Botschaften empfangen worden sind, und fährt fort, *WM_PAINT*-Botschaften an das Programm zu senden.

Sie können im Programmcode sehen, daß *BeginPaint* innerhalb der if-Anweisung steht, so daß es nie aufgerufen wird, wenn der Bildschirm zum erstenmal dargestellt wird. Wenn man die Anweisung *BeginPaint* vor die if-Anweisung stellt, sollte dieses Problem beseitigt sein.

Doch was ist mit der Anweisung *EndPaint*? Sie steht auch innerhalb der if-Anweisung, zusammen mit dem Aufruf von *ReleaseDC*, der *hdc* freigibt; *hdc* ist das Handle des Gerätekontextes, das durch den Aufruf von *BeginPaint* gesetzt wurde. Diese beiden Zeilen sollten nach der if-Anweisung stehen.

Beenden von TDWDEMOA

In TDWDEMOB, der nächsten Programmversion, die Sie sich anschauen, wurde der *WM_PAINT*-Fehler korrigiert. Bevor Sie dieses Programm laden, sollten Sie TDWDEMOA beenden, damit Systemressourcen, die es eventuell belegt, freigegeben werden. Da es sich bei dem einzigen Problem in dem Programm darum handelt, daß der Code innerhalb der if-Anweisung beim ersten Durchlauf nicht ausgeführt wird, sollte das Setzen des Wertes von *CurrentPoint* auf Null eine Ausführung des Codes bewirken und Sie könnten das Programm beenden. Gehen Sie dazu folgendermaßen vor:

1. Löschen Sie im Fenster *Windows Messages* den Namen *WndProc* aus dem Ausschnitt links oben, damit das Programm nicht bei *WM_PAINT* anhält.

2. Drücken Sie F7 bis die folgende Zeile angezeigt wird:

```
if CurrentPoint >= 0
```

3. Markieren Sie *CurrentPoint* und drücken Sie [Ctrl][F4] zum Aufruf des Fensters *Evaluate/Modify*

4. Wählen Sie *Eval*.

5. Positionieren Sie den Cursor in das Feld für den Eintrag des neuen Wertes, geben Sie 0 ein und wählen Sie *Modify* zur Änderung des Variablenwertes. Wenn Sie das Programm jetzt starten, wird die *if*-Anweisung TRUE und die Aufrufe von *BeginPaint* und *EndPaint* werden beim ersten Durchlauf ausgeführt.

6. Drücken Sie F9 zum Start des Programms.

7. Wählen Sie *Quit* zum Verlassen des Programms. Es ist möglich, daß Sie eine Taste drücken müssen, damit der Befehl wirksam wird, weil Windows Maus-Eingabemeldungen von der Systemqueue nicht freigibt. Durch Drücken einer Taste werden alle momentan in der Queue befindlichen Maus-Eingabemeldungen freigegeben.

Debuggen von TDWDEMOB

Nachdem Sie TDWDEMOA beendet haben, können Sie TDWDEMOB laden. Wenn das Programm auf dem Bildschirm erscheint, starten Sie es durch Drücken von F9 und spielen Sie eine Weile damit.

Wenn Sie viele Objekte zeichnen und die Maus besonders oft mit gedrückter linker Taste bewegen, werden Sie bemerken, daß allmählich seltsame Dinge passieren. Zuerst verlängern sich die Antwortzeiten, dann verschwinden Objekte oder Linienfarben und -stärken ändern sich und ein Durcheinander auf dem Bildschirm entsteht. Letztendlich bleibt alles stehen und Sie müssen einen Kaltstart durchführen.

Die wahrscheinlichste Ursache für diese Probleme ist die fehlerhafte Benutzung des Speichers. Um zu bestätigen, daß dies der Fall ist, und um die Fehlersuche zu beginnen, laden Sie Windows erneut, starten wenn möglich nur den Programm-Manager und TDW (zur Reduzierung der Speicherbelegung) und laden TDWDEMOB in TDW.

Das Programm anhalten

Bevor Sie mit dem Debuggen fortfahren, sollten Sie dafür sorgen, daß Sie nicht mit [Ctrl][Alt][SysRq] aus der Anwendung aussteigen müssen, weil Sie TDW sehr vorsichtig behandeln müssen, wenn Sie diese Methode verwenden. Dies erreichen Sie, indem Sie eine einzelne Botschaft bestimmen, bei der das Programm anhalten soll.

Dabei muß es sich um eine Botschaft handeln, die Sie jederzeit während des Programmablaufs produzieren können und die im normalen Betrieb nicht vorkommt.

Weil TDWDEMO ein mausgesteuertes Grafikprogramm ist, das keine Texteingaben annimmt, werden auch keine Tasten verwendet; deshalb ist es eine gute Wahl, das Programm mit Hilfe der Botschaft *WM_KEYDOWN* anzuhalten. Wenn Sie diese Botschaft im Fenster *View/Windows Messages* eingeben, hält das Programm jedesmal an, wenn Sie eine Taste drücken. (Da das Programm auf die rechte Maustaste nicht reagiert, könnten Sie auch WM-RBUTTONDOWN verwenden.)

Testen des Programms

Jetzt können Sie TDWDEMOB mit F9 starten. Bevor Sie irgend etwas in dem Programm tun, prüfen Sie, wieviel Prozent des Systemspeichers für Windows-Programme zur Verfügung stehen, indem Sie in den Programm-Manager gehen, *Hilfe* anwählen und dann *Info über Programm-Manager*. Ein Informationsfenster wird geöffnet, das anzeigt, welche Version von Windows gerade läuft, wieviel freier Speicherplatz verfügbar ist und unten die für Sie interessante Statistik zeigt, nämlich den Prozentsatz freier Systemressourcen.

Gehen Sie wieder in *Simple Paint* und zeichnen Sie. Öffnen Sie nach einer Weile wieder das Fenster *Info über Programm-Manager*. Darin sehen Sie, daß der freie Speicher geschrumpft ist. Wenn Sie in Simple Paint weitermalen, werden die Ressourcen letztendlich auf Null schwinden, und dieselben Effekte wie vorher treten auf.

Was ist zu tun?

Was wissen Sie im Moment? Zum einen muß das Programm globalen Speicher belegen, weil es alle für Windows verfügbaren Systemressourcen verbraucht. Dieses Wissen gibt Ihnen einen Anhaltspunkt, wo Sie ansetzen können: Sie könnten das Programm erneut laden und die Fähigkeit von TDW nutzen, die Belegung des globalen Heap auszugeben. Lassen Sie zunächst eine solche Belegungsliste in einer Log-Datei speichern und dann eine weitere Belegungsliste in einer anderen Log-Datei.

Weil Windows den Inhalt im Speicher bewegt bis die zweite Liste ausgegeben wird, passen die beiden Listen wahrscheinlich nicht zusammen. Um festzustellen, ob sich ein Muster zeigt, müssen Sie die Objekte in der einen Liste einzeln durchgehen und mit den Objekten der anderen Liste, die denselben Eigner haben, vergleichen.

Weil diese Methode sehr langsam ist und die Ergebnisse marginal sind, wird sie nur als Übung für die Durchsicht des globalen Speichers vorgeschlagen. Was Sie tatsächlich tun werden, ist, zu prüfen, ob jedes Speicherobjekt, das vom Programm angelegt wird, auch wieder freigegeben wird.

Vergleichen der Speicher-Belegungslisten

Zur Erzeugung der Belegungslisten gehen Sie wie folgt vor:

1. Starten Sie Windows neu, um sicherzustellen, daß alle von TDWDEMOB belegten Speicherobjekte aus dem globalen Speicher gelöscht werden. Vermeiden Sie wieder das Starten unnötiger Programme, damit die Anzahl von Objekten im globalen Speicher niedrig gehalten wird.

2. Wenn Sie in TDW sind und TDWDEMOB geladen haben, stellen Sie wieder die Programmunterbrechung bei *WM_KEYDOWN* ein.

3. Drücken Sie F9 zum Start des Programms und gehen Sie in TDW zurück, indem Sie eine Taste drücken.

4. Wählen Sie *View/Log* und rufen Sie das lokale Menü auf.

5. Wählen Sie *Open Log File*, geben Sie, wenn sich das Dialogfenster öffnet, einen Namen für die Log-Datei ein und drücken Sie die Eingabetaste ↵.

6. Gehen Sie wieder in das lokale Menü des *Log*-Fensters, wählen Sie *Display Windows Info*, und drücken Sie die Eingabetaste ↵ wenn sich das Dialogfenster *Windows Information* öffnet. Das Drücken der Eingabetaste ↵ bestätigt die Standardeinstellung, mit der Objekte im globalen Speicher ab dem Speicheranfang aufgelistet werden.

7. Wenn TDW mit der Auflistung des globalen Speichers fertig ist, schließen Sie die Datei mit *Close Log File* im lokalen Menü.

8. Löschen Sie das Protokoll mit *Erase Log* im lokalen Menü.

9. Starten Sie das Programm erneut mit F9 und überprüfen Sie mit Hilfe der Maus das Fenster *Info über Programm-Manager*. Beachten Sie den Prozentsatz der verfügbaren Systemressourcen.

10. Zeichnen Sie solange in *Simple Paint*, bis der Prozentsatz der verfügbaren Systemressourcen auf 20 bis 30 % gesunken ist.

11. Gehen Sie durch Drücken einer Taste in TDW zurück und führen Sie die Schritte 4 bis 8 nochmals durch, wobei Sie einen anderen Namen für die Log-Datei verwenden.

12. Verlassen Sie TDW und drucken Sie die Log-Dateien aus. Anschließend können Sie die beiden Ausdrucke vergleichen.

Dabei werden Sie zumindest zwei Dinge feststellen:

- Die Speicherobjekte, die von TDWDEMO angelegt wurden, ändern ihre Größe nicht.
- Die GDI-Speicherobjekte werden größer.

Der erste Umstand weist auf etwas hin, das Sie bereits wissen: fehlerhafter Code in TDWDEMOB belegt globalen und nicht lokalen Speicher.

Der zweite zeigt etwas Neues: TDWDEMOB belegt Objekte des Graphics Device Interface und gibt Sie nicht wieder frei.

Auffinden des Fehlers, ein funktionaler Lösungsansatz

Da Sie jetzt eine Vorstellung von dem Fehler haben, können Sie damit beginnen, das Programm nach Stellen zu durchsuchen, wo Speicherobjekte belegt und nicht wieder freigegeben werden. Ein nützlicher Ansatz ist, sich eine Übersicht über die Funktionen des zu untersuchenden Programms zu verschaffen und dabei jede Routine zu untersuchen.

Auswahl von Menüpunkten

Menüpunkte werden ausgewählt, indem Sie den Cursor auf das Menü positionieren, den linken Mausknopf drücken, und das Menü herunterziehen, bis Sie eine Auswahl getroffen haben, die zum Beispiel die Farbe, Zeichenstiftdicke oder Form ändert. Bei Änderung einer dieser Einstellungen, sendet Windows die Botschaft *WM_COMMAND* an die Routine *WndProc*, die auf die Botschaft mit dem Aufruf von *DoWMCommand* reagiert.

DoWMCommand ist eine Switch-Anweisung, die Ihre Auswahl in einer Programmvariablen ablegt. Diese Variablen werden im Datensegment von TDWDEMO gespeichert und haben keinen Einfluß auf den globalen Speicher.

Zeichnen von Formen

Formen zeichnen Sie, indem Sie den Cursor im Client-Bereich des Fensters positionieren, die linke Maustaste gedrückt halten, die Maus bewegen und die Maustaste wieder loslassen. Wenn Sie die Maus mit gedrückter linker Taste herumbewegen, bemerken Sie, daß die Form während der Mausbewegung gezeichnet, gelöscht und wieder gezeichnet wird. Die Form wird erst dann auf dem Bildschirm fixiert, wenn Sie die linke Maustaste loslassen.

Drücken der linken Maustaste

Wenn Sie im Client-Bereich die linke Maustaste drücken, sendet Windows die Botschaft *WM_LBUTTONDOWN* an *WndProc*, die *DoButtonDown* aufruft. Diese Routine speichert die aktuelle Mausposition (im folgenden Anker genannt) und Zeichenstifteinstellungen in der Struktur *thisShape*. Diese Struktur ist eine Programmvariable und betrifft nur das Datensegment von TDWDEMO.

Bewegen der Maus

Wenn Sie die Maus mit gedrückter linker Maustaste bewegen, sendet Windows die Botschaft *WM_MOUSEMOVE* (oder die identische Botschaft *WM_MOUSEFIRST*) an *WndProc*, die *DoMouseMove* aufruft. Diese Routine verwendet *DrawShape* zum Löschen der Form von der vorherigen Mausposition zurück zur Ursprungsposition und ruft dann wieder *DrawShape* auf, um die Form von der aktuellen Position aus zum Anker zu zeichnen. *DoMouseMove* macht von dem globalen Speicher nur Gebrauch, um den Gerätekontext des aktuellen Fensters zu erhalten. Dieser Gerätekontext wird am Ende der Routine mit dem Aufruf von *ReleaseDC* wieder freigegeben.

Zeichnen einer Form (und Auffinden des Fehlers)

Die nächste Routine, die wir näher betrachten wollen, ist *DrawShape*, die zweimal von *DoMouseMove* aufgerufen wird. *DrawShape* speichert den Zeichenstift mit dem die vorherige Form gezeichnet wurde, erzeugt einen neuen Stift und zeichnet dann eine Linie, eine Ellipse oder ein Rechteck. Das letzte was sie tut, ist das Wiederherstellen des Stifts, den sie am Anfang der Routine gespeichert hat.

Da ein Zeichenstift ein Objekt des Graphics Device Interface (GDI) ist, das im globalen Speicher abgelegt wird, könnte *DrawShape* Code enthalten, der für die Speicherprobleme verantwortlich ist. Diese Routine ist besonders verdächtig, weil sie bei jeder Mausbewegung zweimal aufgerufen wird. Wenn sie Stifte erzeugt und nicht wieder löscht, wird sie sehr schnell den Speicher aufbrauchen.

Tatsächlich belegt *DrawShape* ziemlich am Anfang mit einem Aufruf von *SelectObject* ein Objekt ohne es am Ende mit dem Aufruf von *DeleteObject* wieder freizugeben. Um diesen Fehler zu beheben, müssen Sie die letzte Zeile in *DrawShape* durch folgende Anweisung ersetzen:

```
DeleteObject(SelectObject(handleDC,saveObjectt));
```

Es ist möglich, daß dies die einzige Ursache für die Speicherprobleme war. Aber um sicherzugehen, sollten Sie sich den Rest des Zeichenprozesses ansehen.

Loslassen der linken Maustaste

Wenn Sie die Maustaste loslassen, zeichnet TDWDEMO die Figur zum letzten Mal und läßt sie auf dem Bildschirm stehen. Beim Loslassen der Taste sendet Windows die Botschaft *WM_LBUTTONUP* an *WndProc*, die *DoLButtonUp* aufruft. Diese Routine sichert das aktuelle Rechteck im Benutzerbereich in die aktuelle Form im Array *thisShape*, ruft *InvalidateRect* auf, um den Bereich zum Update-Bereich des Fensters hinzuzufügen und ruft dann *UpdateWindow* auf, die die Botschaft *WM_PAINT* direkt an das Hauptfenster schickt. Diese Routine verwendet keinen globalen Speicher.

Wenn die Routine *DoLButtonUp* beendet ist, steht in der Queue eine *WM_PAINT*-Botschaft, die von *WndProc* bearbeitet werden kann.

Malen des Bildschirms

Wenn WndProc die Botschaft *WM_PAINT* erhält, wird *DoPaint* aufgerufen, die den entsprechenden Bereich des Bildschirms neu malt. Beim Neumalen des

Update-Rechtecks ruft *DoPaint* zwei Windows-Routinen auf, die auf den globalen Speicher zugreifen: *CreateCompatibleDC* und *SelectObject*. Am Ende von *DoPaint* werden *DelecteDC* und *DeleteObject* aufgerufen, wodurch der am Anfang der Routine belegte globale Speicher wieder freigegeben wird.

Anmerkung

Da Sie nun jede Routine des Programms untersucht haben, können Sie ziemlich sicher sein, daß Sie den Speicherplatz-Fehler gefunden haben. Diese Methode wurde gewählt, weil Sie mit dem Programm nicht vertraut sind. Wenn Sie das Programm selbst geschrieben hätten, hätten Sie natürlich die Speicherbelegungsanweisungen viel schneller gefunden.

Es hätte auch geholfen, mehr zu testen. Mit genügend Tests und Kenntnis des Programms hätten Sie den Teil der Fehlersuche, der die Menüs betraf, weglassen können, weil Sie den Fehler einfach durch Bewegen der Maus im Bildschirm erzeugen konnten. Wenn Sie darüber hinaus gewußt hätten, daß das Problem auch auftritt, wenn Sie einfach nur die Maus im Client-Bereich mit gedrückter linker Taste bewegen, (das hätten Sie feststellen können, indem Sie die Maus mit gedrückter Taste bewegen bis sich das Programm aufhängt), hätten Sie *DoMouseMove* schnell als die Routine identifizieren können, auf die Sie Ihre Anstrengungen konzentrieren müssen.

A. Kommandozeilen-optionen

Beim Aufruf von Turbo Debugger können zusätzliche Optionen in der Kommandozeile angeben werden. Die allgemeine Syntax des Aufrufs lautet:

```
td [ Optionen ] [ Programmname [ Programmargumente ] ]
```

Optionale Eingaben stehen in eckigen Klammern. Ein Minuszeichen hinter einer Option schaltet die entsprechende Option ab.

Option	Beschreibung
-c*Dateiname*	Konfigurationsdatei
-do	Debugger verwendet den zweiten Bildschirm
-ds	Bildschirmdarstellung von Programm und Debugger wird in den Programmspeicher ausgelagert
-h, -?	Hilfsinformationen für alle Kommandozeilenoptionen ausgeben
-l	Starten mit Assemblerdarstellung
-p	Mausunterstützung,
-sc	Keine Unterscheidung zwischen Groß- und Kleinschreibung
-sd*Verzeichnis*	Verzeichnisse für Quelldateien
-t*Verzeichnis*	Startverzeichnis

B. Inline-Assemblerbefehle

In diesem Anhang sind die Befehlskurzformen und andere spezielle Symbole zur Eingabe von Befehlen mit dem Inline-Assembler aufgelistet. Es werden dieselben Kurzformen wie in Turbo Assembler verwendet.

Tabelle B.1 8086- / 80186- und 80286-Befehle

AAA	INC	LIDT**	REPNZ
AAD	INSB*	LLDT**	REPZ
AAM	INSW*	LMSW**	RET
AAS	INT	LOCK	RETF
ADC	INTO	LODSB	ROL
ADD	IRET	LODSW	ROR
AND	JB	LOOP	SAHF
ARPL**	JBE	LOOPNZ	SAR
BOUND*	JCXZ	LOOPZ	SBB
CALL	JE	LSL**	SCASB
CLC	JL	LTR**	SCASW
CLD	JLE	MOV	SGDT**
CLI	JMP	MOVSB	SHL
CLTS**	JNB	MOVSW	SHR
CMC	JNBE	MUL	SLDT**
CMP	JNE	NEG	SMSW**
CMPSB	JNLE	NOP	STC
CMPSW	JNO	NOT	STD
CWD	JNP	OR	STI
DAA	JO	OUT	STOSB
DAS	JP	OUTSB*	STOSW
DEC	JS	OUTSW*	STR**
DIV	LAHF	POP	SUB
ENTER*	LAR**	POPA*	TEST
ESC	LDS	POPF	WAIT
HLT	LEA	PUSH	VERR**
IDIV	LEAVE*	PUSHA*	VERW**
IMUL	LES	PUSHF	XCHG
IN	LGDT**	RCL	XLAT
			XOR

* nur ab 80186- und 80286-Prozessor verfügbar
** nur ab 80286-Prozessor verfügbar

Tabelle B.2 80386-Befehle

BSF	LSS	SETG	SETS
BSR	MOVSX	SETL	SHLD
BT	MOVZX	SETLE	SHRD
BTC	POPAD	SETNB	CMPSD
BTR	POPFD	SETNE	STOSD
BTS	PUSHAD	SETNL	LODSD
CDQ	PUSHFD	SETNO	MOVSD
CWDE	SETA	SETNP	SCASD
IRETD	SETB	SETNS	INSD
LFS	SETBE	SETO	OUTSD
LGS	SETE	SETP	JECXZ

Tabelle B.3 80486-Befehle

BSWAP	INVLPG
CMPXCHG	WBPINVD
INVD	XADD

Tabelle B.4 CPU-Register

Byte-Register	ah, al, bh, bl, ch, cl, dh, dl
Wort-Register	ax, bx, cx, dx, si, di, sp, bp
Segmentregister	cs, ds, es, ss
Fließkommaregister	st, st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7)

Tabelle B.5 80386-/80486-Register

EAX	EDI
EBX	EBP
ECX	ESP
EDX	FS
ESI	GS

Tabelle B.6 Reservierte Bezeichner

WORD PTR	TBYTE PTR
BYTE PTR	NEAR
DWORD PTR	FAR
QWORD PTR	SHORT

Tabelle B.7 8087- und 80287-Befehle

FABS	FIADD	FLDL2E	FST
FADD	FICOM	FLDL2T	FSTCW
FADDP	FICOMP	FLDPI	FSTENV
FBLD	FIDIV	FLDZ	FSTP
FBSTP	FIDIVR	FLD1	FSTSW*
FCHS	FILD	FMUL	FSUB
FCLEX	FIMUL	FMULP	FSUBP
FCOM	FINCSTP	FNOP	FSUBR
FCOMP	FINIT	FNSTSW**	FSUBRP
FCOMPP	FIST	FPATAN	FTST
FDECSTP	FISTP	FPREM	FWAIT
FDISI	FISUB	FPTAN	FXAM
FDIV	FISUBR	FRNDINT	FXCH
FDIVP	FLD	FRSTOR	FXTRACT
FDIVR	FLDCW	FSAVE	FYL2X
FDIVRP	FLDENV	FSCALE	FYL2XP1
FENI	FLDLG2	FSETPM*	F2XM1
FFREE	FLDLN2	FSQRT	

* nur ab dem 80287-Koprozessor verfügbar
** ab dem 80287-Koprozessor kann man bei den Befehlen FSTSW und FNSTSW auch das AX-Register zum Übertragen des Statusworts angeben

Tabelle B.8 80387-/80487-Befehle

FCOS	FUCOM
FSIN	FUCOMP
FPREM1	FUCOMPP
FSINCOS	

C. Die Installation von TDW

Turbo Debugger ist startbereit, sobald das Installationsprogramm die Dateien auf Ihre Festplatte kopiert und eine eigene Programmgruppe unter Windows erzeugt hat.

Mit Hilfe des Programms TDWINST.EXE haben Sie allerdings die Möglichkeit, eine Anzahl von Standardoptionen zu verändern. Auch durch Angabe von Optionen in der Kommandozeile beim Start von Turbo Debugger können Sie einige Einstellungen des Debuggers Ihrer Verwendung gemäß ändern. Sollten Sie dabei feststellen, daß Sie manche Kommandozeilenoptionen immer wieder benötigen, können Sie diese mit Hilfe des Installationsprogramms fest abspeichern. Folgende Einstellungen lassen sich mit dem Installationsprogramm verändern:

- Farben der Fenster, Dialogfenster und Menüs.
- Bildschirmparameter: Auslagerung des Bildschirms, Integerdarstellung, Startbildschirm (Quell- oder Assemblercode), Anzahl der Bildschirmzeilen, Tabulatoreinstellung, maximale Größe nicht-überlappender *Watches*-Fenster, 43-/50-Zeilenmodus und Umfang des Protokolls.
- Suchpfade der Quelldateien sowie Hilfe- und Konfigurationsdateien.
- Eingabe- und Tastaturparameter: Ctrl-Tastenkürzel, Länge der History-Liste sowie Aufnahme von Tastaturbefehlen.
- Quellcode-Optionen: Programmiersprachen und Unterscheidung zwischen Groß- und Kleinschreibung.
- Bildschirm-Modus.

Der Aufruf von TDWINST

Um das Installationsprogramm zu starten geben Sie entweder auf DOS-Ebene TDWINST ein oder benutzen den Befehl *Datei/Ausführen...*im Programm-Manager. Danach können Sie entweder einen der unterlegten Anfangsbuchstaben der Menüoptionen eingeben oder mit Hilfe der beiden Pfeiltasten, ↑ und ↓, den gewünschten Menüpunkt ansteuern und dann die Eingabetaste ↵ drücken. Mit

der Eingabe von D können Sie zum Beispiel die Bildschirmdarstellung ändern. Die anderen Menüs des Installationsprogramms werden ebenso bedient. Um zum übergeordneten Menü zurückzukommen, drücken Sie die Taste Esc und wiederholen dies gegebenenfalls, bis das Hauptmenü wieder aktiv ist.

Die Einstellung der Bildschirmfarben

Durch die Auswahl von *Colors* erhalten Sie das *Colors*-Menü. Nun haben Sie die Möglichkeit, zwischen zwei Menüpunkten zu wählen. *Customize* (engl. individuell gestalten) und *Default color set* (engl. Standardfarbpalette).

Farben anpassen

Wenn Sie *Customize* auswählen, erscheint ein drittes Menü, das Ihnen erlaubt, die Farbgestaltung von Dialogfenstern, Menüs usw. zu verändern.

Windows

Für die Anpassung von Fenstern wählen Sie den Befehl *Windows*. Dadurch wird ein viertes Fenster geöffnet, in dem Sie den Fenstertyp auswählen können, der verändert werden soll: *Text*, *Data*, *Low Level* (in diesem Fall das *CPU*-Fenster) und *Other* (zum Beispiel das *Breakpoints*-Fenster). Durch die Auswahl einer dieser Optionen erhalten Sie ein weiteres Fenster mit den Fensterelementen und einer Darstellung von Beispielfenstern (ein aktives und ein nicht aktives), mit der Sie die verschiedenen Farbkombinationen testen können. Der Bildschirm sieht wie in Abbildung C.1 dargestellt, aus:

Wenn Sie ausgewählt haben, was Sie verändern möchten, erscheint ein Kästchen mit der Farbpalette über dem Menü. Mit Hilfe der Pfeiltasten können Sie sich innerhalb des Palettenkästchens bewegen. Rechts davon sehen Sie alle

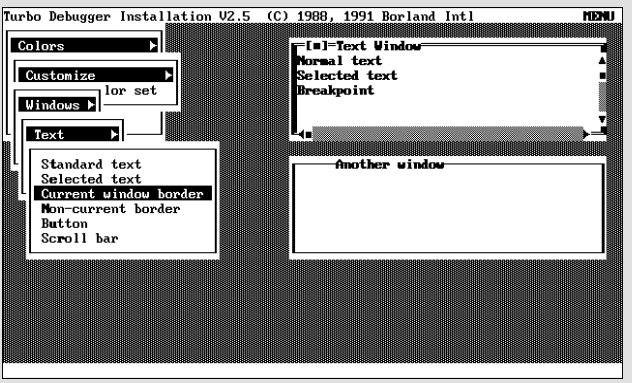


Abbildung C.1 Die Einstellung der Bildschirmfarben der (ein-

Fenster oder Kästchen in der Kategorie, die Sie verändern. Wenn Sie nun den Cursor durch die Farbauswahl bewegen, so erscheint das Fenster, das Sie verändern wollen, in der jeweiligen Farbe, damit Sie sehen können, ob Ihnen die Farbeinstellung zusagt. Wenn Sie die richtige Einstellung gefunden haben, drücken Sie die Eingabetaste, um den Vorgang abzuschließen.

Achtung: Turbo Debugger verwaltet drei Farbtabellen: für Farb-, Schwarzweiß- und Monochrombildschirme. Sie können davon nicht mehrere gleichzeitig verändern, sondern immer nur die Tabelle, die von Ihrem aktuellen Bildschirmmodus und der Art Ihres Bildschirms abhängt. Wenn Sie also mit einem Farbbildschirm arbeiten und die Schwarzweißtabelle verändern möchten, müssen Sie zuerst Ihren Bildschirmmodus auf der DOS-Ebene (mode bw80) ändern und dann TDWINST erneut starten.

Dialog boxes

Wenn Sie im Menü *Custumize* den Menüpunkt *Dialogs* ausgewählt haben, erscheint eine Liste mit Dialogfenstern und Menüpunkten sowie ein Dialogfenster, mit dem Sie experimentieren können.

Der Bildschirm sieht wie folgt aus:

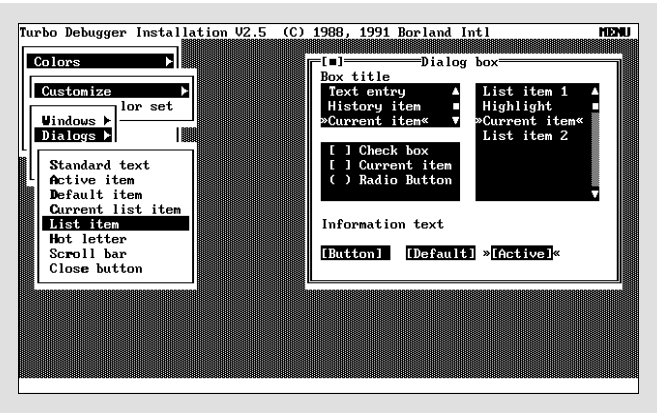


Abbildung C.2 Einstellungen für Dialogfenster

Wie im *Windows*-Menü wird durch die Auswahl eines Menüpunkts eine Palette geöffnet, aus der Sie die gewünschte Farbe für dieses Element auswählen können.

Menus

Nach Auswahl von *Menus* erscheint ein Menü mit den verfügbaren Menüpunkten zusammen mit einem Testmenü. Wählt man ein Element in diesem Menü aus, so erscheint die gewohnte Farbpalette.

Screen

Durch die Auswahl von *Screen* erscheint ein Menü, aus dem Sie auf ein weiteres Menü zugreifen können, in dem Bildschirmmuster und Farbpaletten

für die einzelnen Bildschirmelemente enthalten sind. Außerdem erscheint ein Testhintergrund, mit dem Sie die Auswahl überprüfen können.

Die Standard-farben

Wenn Sie im *Colors*-Menü den Punkt *Default color set* auswählen, erscheint ein aktives Fenster und ein nicht aktives Fenster auf dem Bildschirm, in denen Sie die Standardfarben für die einzelnen Elemente überprüfen können.

Die Einstellung der Bildschirmparameter in Turbo Debugger

Wenn Sie im Hauptmenü die Option *Display* auswählen, erhalten Sie das Dialogfenster *Display options*.

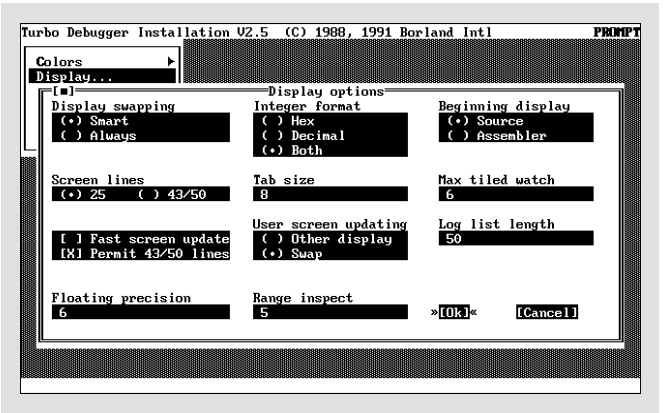


Abbildung C.3 Das Dialog-fenster Display options

Hinweis: Einige der dargestellten Optionen können Sie sowohl mit TDWINST fest einstellen als auch in der Kommandozeile von Turbo Debugger angeben. Am Ende dieses Anhangs finden Sie eine Übersicht über alle Kommandozeilenoptionen und die entsprechenden Einstellungen in TDWINST.

Display swapping

Mit den Auswahlfeldern unter *Display swapping* legen Sie fest, wie Turbo Debugger zwischen seinem eigenen Bildschirm und dem Programmbildschirm umschaltet. Folgende Einstellungen können ausgewählt werden:

Smart Schaltet Sie nur dann auf den Benutzerbildschirm um, wenn

eventuell etwas ausgegeben wird. Turbo Debugger schaltet jedesmal um, wenn Sie bei der Einzelschrittausführung zu einer Routine kommen oder eine Zeile ausführen, in der etwas auf dem Bildschirm ausgegeben wird. Das ist die Standardeinstellung.

Always Schaltet jedesmal um, wenn das Benutzerprogramm läuft. Verwenden Sie diese Option, wenn Turbo Debugger im Smart-Modus nicht alle Ausgaben erwischt. Wenn Sie diese Option verwenden, flackert der Bildschirm im Einzelschrittverfahren bei jedem Schritt auf, da der Bildschirm von Turbo Debugger mit dem Programmbildschirm ausgetauscht werden muß.

Integer format

Mit den Auswahlfeldern unter *Integer format* können Sie bestimmen, wie Integer-Zahlen dargestellt werden sollen. Sie können folgende Einstellungen vornehmen.

Hex für eine Darstellung als Hexadezimalzahlen.

Decimal für eine Darstellung als Dezimalzahlen.

Both für eine Darstellung beider Systeme.

Beginning Display

Mit den Auswahlfeldern unter *Beginning display* legen Sie fest, in welcher Sprache Ihr Programm nach dem Aufruf dargestellt wird. Zwei Einstellungen sind möglich:

Assembler Assemblerstart: Keines Ihrer Programme wird ausgeführt. Ein *CPU*-Fenster zeigt den ersten Befehl in Ihrem Programm an.

Source Quellcodestart: Der Initialisierungscode Ihres Compilers wird abgearbeitet. Sie beginnen Ihre Arbeit in einem *Module*-Fenster, in dem das Hauptprogramm dargestellt ist.

Screen lines

Mit Hilfe dieser Auswahlfelder schalten Sie zwischen dem 25- und 43/50-Zeilenmodus um.

Achtung: Nur mit EGA- oder VGA-Ausrüstung lassen sich mehr als 25 Zeilen darstellen.

Tab size

In diesem Eingabefeld wird die Spaltenbreite der Tabulatoren in Text- oder Dateifenstern festgelegt. Sie werden nach der Anzahl der Spalten gefragt (die Anzahl muß zwischen 1 und 32 liegen); die Vorgabe ist 8.

Maximum tiled watch

In diesem Eingabefeld geben Sie an, auf wieviele Zeilen ein *Watches*-Fenster im nicht überlappenden Modus maximal ausgedehnt werden kann. Zulässig ist eine Eingabe von 1 bis 20 Zeilen.

Fast screen update

Im Markierungsfeld *Fast screen update* können Sie Ihr System auf »schnelle« Bildschirmausgabe einstellen. Schalten Sie diese Option aus, wenn es bei der schnellen Ausgabe zu Bildschirmstörungen (»Schnee«) kommt.

Permit 43/50 line mode

In diesem Markierungsfeld legen Sie fest, ob der 43-Zeilen-Modus oder der 50-Zeilen-Modus zugelassen wird. Falls Sie diese Option ausschließen, können Sie etwa 8 KB an Speicherplatz einsparen. Wenn Sie diese Option abgeschaltet haben, können Sie den Bildschirm nicht mehr in den 43/50-Zeilenmodus umschalten, auch wenn Ihr System dazu in der Lage wäre.

User screen updating

Mit den Auswahlfeldern unter *User screen updating* stellen Sie ein, auf welche Weise der Bildschirm aktualisiert wird, wenn zwischen der Darstellung Ihres Programms und der Darstellung von Turbo Debugger hin- und hergeschaltet wird. Mit *User screen updating* können Sie zwischen folgenden Optionen umschalten:

Other display Turbo Debugger wird auf einem zweiten angeschlossenen

Bildschirm dargestellt. Wenn Sie sowohl einen Farb- als auch einen Monochrombildschirmadapter haben, können Sie mit dieser Option Ihren Programmbildschirm auf dem einen Bildschirm und Turbo Debugger auf dem anderen gleichzeitig darstellen.

Swap Benützt einen einzigen Bildschirmadapter und eine Bildschirmoberfläche. Die Inhalte des Benutzerbildschirms und des Turbo-Debugger-Bildschirms werden immer zwischen Hauptspeicher und Bildschirmadapter hin- und hergeladen. Dies ist die langsamste Methode des Bildschirmwechsels, aber auch die sicherste und diejenige, die das Benutzerprogramm am geringsten beeinflußt. Wählen Sie diese Option, wenn Sie ein Programm mit mehreren logischen Bildschirmseiten debuggen.

Log list length

In diesem Eingabefeld legen Sie fest, wieviele Ereignisse im Protokoll zwischengespeichert werden sollen. Zulässig sind Werte zwischen 4 und 200 Einträgen.

Floating precision

In diesem Eingabefeld legen Sie fest, mit wievielen Nachkommastellen eine Gleitkommazahl dargestellt werden soll. Zulässig sind Werte zwischen 1 und 32 Stellen.

Range inspect

In diesem Eingabefeld legen Sie die ahl der Elemente fest, die TDW für einen Zeiger ausgibt, nachdem Sie den Befehl *Range* im lokalen Menü eines Inspektor-Fensters gewählt haben. Gemäß Voreinstellung gibt TDW nur die ersten fünf Elemente aus, die über den Zeiger referenziert werden, es sei denn, Sie hätten über den lokalen Befehl *Range* eine andere Anzahl angegeben. Der größte Wert, den Sie hier eingeben können, ist 4096.

Die Einstellung der Turbo-Debugger-Optionen

Durch die Auswahl des Befehl *Options* erhalten Sie ein Menü mit Optionen, die wiederum zu Dialogfenstern führen.

Directories

In diesem Dialogfenster befinden sich zwei Eingabefelder, in denen folgende Verzeichnispfade angegeben werden können:

Source directories gibt die Verzeichnisse an, in denen Turbo Debugger nach Quelldateien sucht.

Turbo directory enthält das Verzeichnis, in dem Turbo Debugger nach der Hilfedatei und der Konfigurationsdatei suchen soll.

Input & Prompting...

In diesem Dialogfenster können Sie einstellen, wie Eingaben in Turbo Debugger verarbeitet und entgegengenommen werden sollen:

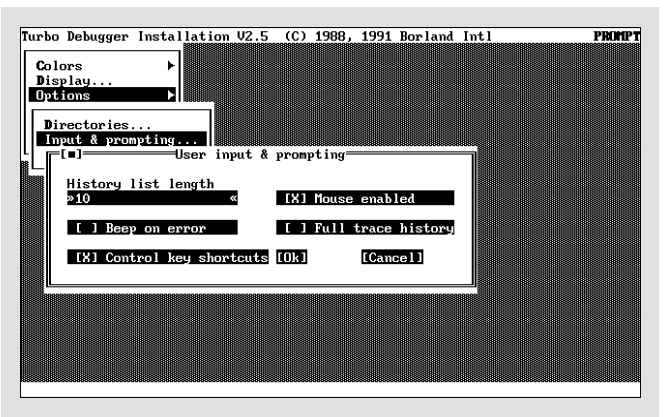


Abbildung C.4 Die Dialogbox user input & prompting

History list length
In diesem Eingabefeld können Sie angeben, wieviele Einträge in einer History-Liste erhalten bleiben sollen.

Beep on error
Standardmäßig gibt Turbo Debugger einen Alarmton aus, wenn Sie eine

unzulässige Taste drücken oder eine Fehlermeldung verursachen. Mit dem Markierungsfeld *Beep on error* können Sie diese Voreinstellung verändern.

Control key shortcut
Dieses Markierungsfeld schaltet die Ctrl-Kurzcodes ein oder aus. Wenn die Ctrl-Kurzcodes eingeschaltet sind, kann man jeden Befehl in einem lokalen Menü direkt mit der Ctrl-Taste und dem ersten Buchstaben des Menüpunkts aufrufen. Man kann dann aber die WordStar-ähnlichen Befehle für die Cursorbewegungen nicht verwenden.

Mouse enabled
Dieses Markierungsfeld legt fest, ob Turbo Debugger standardmäßig eine Maus unterstützt.

Full trace history
Aktivieren Sie diese Option, wenn TDW ein Protokoll Ihrer Debugger-Sitzung mitführen soll. Ein solches Protokoll erlaubt Ihnen, Ihr Programm »rückwärts« ausführen zu lassen, verlangsamt aber die normale Ausführung des Programms erheblich.

Source Debugging...

Im Dialogfenster *Source Debugging* legen Sie fest, welche Programmiersprache für die Auswertung von Ausdrücken verwendet und ob zwischen Groß- und Kleinschreibung unterschieden werden soll.

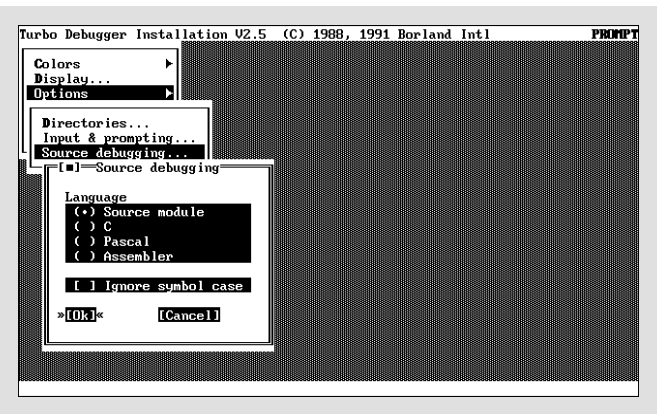


Abbildung C.5 Die Dialogbox Source Debugging

Language
In den Auswahlfeldern *Language* kann festgelegt werden, in welcher Sprache Turbo Debugger Ausdrücke auswerten soll. Sie haben die Auswahl zwischen:

Source module	Die Programmiersprache wird anhand der Sprache des aktuellen Quellcodes ausgewählt.
C	Verwendet ausschließlich C-Ausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.
Pascal	Verwendet ausschließlich Pascal-Ausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.
Assembler	Verwendet ausschließlich Assemblerausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.

Ignore symbol case
Wenn dieses Feld markiert ist, wird zwischen Groß- und Kleinschreibung nicht unterschieden. Dies kann man ändern, indem man die Markierung in diesem Feld aufhebt.

Das Einstellen des Bildschirmmodus

Unter diesem Menüpunkt legen Sie den Bildschirmmodus fest.

Default
Turbo Debugger stellt die Art Ihre Bildschirmadapters fest und wählt den Modus aus, der hierfür am besten geeignet ist.

Color
Sie erhalten eine farbige Darstellung, wenn Sie einen der folgenden Grafikadapter verwenden: CGA, MCGA, EGA, VGA oder 8514 von IBM.

Black & White
Sie erhalten Schwarzweiß-Darstellung, wenn Sie einen der folgenden Grafikadapter verwenden: CGA, MCGA, EGA, VGA oder 8514 von IBM.

Monochrome
Wählen Sie diese Einstellung, wenn Sie einen Farbmonitor zusammen mit einer Herculeskarte oder einer Monochrome-Karte verwenden.

LCD
Wählen Sie diese Einstellung, wenn Sie einen LCD-Monitor verwenden.

Kommandozeilenoptionen und Installationsoptionen

Einige der Optionen, die im vorliegenden Abschnitt beschrieben wurden,

kann man auch als Kommandozeilenoptionen beim Aufruf von Turbo Debugger angeben. Die folgende Tabelle zeigt Entsprechungen zwischen den Kommandozeilenoptionen des Debuggers und den Befehlen des Installationsprogramms, wodurch man die Optionen dauerhaft abspeichern kann.

Tabelle C.1 Kommandozeilenoptionen von Turbo Debugger

Option	TDWINST-Menüauswahl	Dialogfenster
	Display	*Display Options*
-do		(•) Other Display
-ds		(•) Swap
	Display	*Display Options*
-l		(•) Assembler
-l-		(•) Source
	Options / Input & Prompting	*User Input & Prompting*
-p		[X] Mouse enabled
-p-		[ ] Mouse enabled
	Options / Directories	*Directories*
-sd		Source Directories
-t		Turbo Directory

Hinweis: Mit den Optionen -h und -? erhalten Sie eine Liste aller verfügbaren Kommandozeilenoptionen für TDWINST.EXE.

Wenn alles wie gewünscht eingestellt wurde

Speichern der Änderungen

Nachdem Sie alle Optionen von Turbo Debugger wie gewünscht eingestellt haben, wählen Sie den Befehl *Save* des Hauptmenüs aus, um festzulegen, wie die Konfiguration gespeichert werden soll.

Save Configuration File
Wenn Sie *Save Configuration File* wählen, wird ein Dialogfenster geöffnet, deren Eingabefeld mit dem Standarddateinamen TDCONFIG.TDW initialisiert wird. Sie können diesen Namen mit [¶] übernehmen oder einen anderen Namen für die Konfigurationsdatei eingeben. Falls Sie einen anderen Dateinamen angeben, können Sie die Datei beim Start von Turbo Debugger mit der Kommandozeilenoption -c laden. Zum Beispiel:

```
tdw -cspezial.tdw prox
```

Die Datei kann aber auch mit dem Befehl *Options/Restore configuration* während der Arbeit mit Turbo Debugger eingelesen werden.

Modify TDW.EXE

Wählen Sie *Modify TDW.EXE* aus, werden alle Konfigurationsänderungen direkt in der Programmdatei von Turbo Debugger, TDW.EXE, gespeichert. Beim nächsten Aufruf von Turbo Debugger werden diese Einstellungen standardmäßig vorgegeben.

TDWINST verlassen

Mit dem Befehl *Quit* des Hauptmenüs können Sie TDWINST jederzeit verlassen.

D. Dialogfenster und Fehlermeldungen

Turbo Debugger gibt Fehlermeldungen und Dialogfenster an der aktuellen Cursor-Position aus. In diesem Kapitel werden alle Dialogfenster und Fehlermeldungen beschrieben, die von Turbo Debugger erzeugt werden.

Hier wird beschrieben, wie Sie auf Dialogfenster oder Fehlermeldungen reagieren können. Alle Dialogfenster und Fehlermeldungen (inklusive der schwerwiegenden Fehler beim Programmstart) sind in alphabetischer Reihenfolge aufgeführt und werden einzeln beschrieben.

Dialogfenster

Turbo Debugger liefert ein Dialogfenster, wenn zusätzliche Informationen benötigt werden, um einen Befehl abzuschließen. Die Überschrift des Dialogfensters beschreibt kurz, welche Informationen eingegeben werden sollen. In manchen Dialogfenstern steht eine History-Liste mit bereits früher eingegebenen Antworten zur Verfügung.

Ein Dialogfenster kann auf zwei Weisen bearbeitet werden:

- Sie geben die Antwort ein und drücken ↵.
- Sie drücken Esc, um den Dialog abzubrechen und zu dem Menübefehl zurückzukehren, mit dem das Dialogfenster aufgerufen wurde.

In manchen Dialogfenstern haben Sie genau zwei Antwortmöglichkeiten (wie *Yes*/*No*). Mit Hilfe der Tab-Taste können Sie zwischen den beiden Möglichkeiten umschalten. Anschließend treffen Sie Ihre Auswahl mit ↵ oder wählen direkt Y oder N. Mit Hilfe von Esc kann die Ausführung des Befehls sofort beendet werden.

Eine vollständige Besprechung der Tastaturbefehle in Dialogfenstern finden sie in Kapitel 2 »Fehlersuche mit Turbo Debugger«.

Hier nun eine alphabetische Auflistung aller Meldungen, die von Dialogfenstern erzeugt werden:

Device error – Retry?
[Gerätefehler – Noch einmal probieren?]
Beim Schreiben auf ein Zeichenausgabegerät, zum Beispiel auf den Drucker, ist ein Fehler aufgetreten. Die Ursache kann entweder darin liegen, daß der Drucker nicht angeschlossen ist oder kein Papier mehr hat. Korrigieren Sie den Sachverhalt und geben Sie dann Y ein, um es nochmal zu versuchen, oder brechen Sie den Vorgang durch Eingabe von N ab.

Disk error on drive ___ – Retry?
[Laufwerksfehler auf Laufwerk ___ – Noch einmal probieren?]
Beim Zugriff auf das angegebene Laufwerk ist ein schwerwiegender Fehler aufgetreten. Das kann heißen, daß Sie entweder keine Diskette im Laufwerk haben, oder, falls eine Festplatte vorhanden ist, daß die Festplatte einen nicht mehr lesbaren oder beschreibbaren Sektor aufweist. Sie können Y eingeben, um zu sehen, ob ein zweiter Versuch zum Erfolg führt, sonst geben Sie N ein, um den Vorgang abzubrechen.

Edit watch expression
[Watch-Ausdruck editieren]
Sie müssen den Ausdruck im *Watches*-Fenster ändern oder ersetzen. Das Dialogfenster wird mit dem hervorgehobenen Watch-Ausdruck initialisiert.

Enter address, count, byte value
[Adresse, Anzahl und Initialisierungswert eingeben]
Geben Sie die Adresse des Speicherblocks ein, den Sie mit einem bestimmten Wert initialisieren möchten. Anschließend geben Sie die Anzahl der Bytes ein, die Sie initialisieren wollen. Als drittes Argument muß der Wert angegeben werden, mit dem Sie den Speicherbereich füllen wollen.

Enter adress to position to
[Geben Sie die Adresse ein, zu der Sie springen wollen]
Geben Sie die Adresse ein, die Sie in Ihrem Programm sehen wollen. Sie können einen Funktionsnamen, eine Zeilennummer, eine absolute Adresse oder einen Zeigerausdruck eingeben. In Kapitel 9 finden Sie weitere Informationen über die Eingabemöglichkeiten von Adressen.

Enter animate delay (10ths of sec)
[Geben Sie die Zeitspanne für die Verzögerung (in Zehntelsekunden) in der Animation ein]
Geben Sie an, wie schnell der Befehl *Animate* ausgeführt werden soll. Je höher die eingegebene Zahl ist, desto länger ist die Zeitspanne zwischen zwei aufeinanderfolgenden Schritten während der Animation.

Enter code address to execute to
[Geben Sie die Codeadresse ein, bis zu der das Programm laufen soll]
Geben Sie die Adresse in Ihrem Programm ein, an der das Programm unterbrochen werden soll. In Kapitel 9 »Ausdrücke« finden Sie eine Erläuterung über die Möglichkeiten der Adreßeingabe.

Enter command line arguments
[Geben Sie die Kommandozeilenargumente ein]
Geben Sie die Kommandozeilenargumente für das Programm ein, das Sie debuggen wollen.

Enter comment to add to end of log
[Geben Sie einen Kommentar ein, der am Ende des Protokolls angefügt werden soll]
Geben Sie eine beliebige Textzeile ein, die im *Log*-Fenster angehängt werden soll. Die Zeile wird dem Protokoll genau wie eingegeben angefügt.

Enter expression for conditional breakpoint
[Geben Sie einen Ausdruck für einen bedingten Breakpoint ein]
Geben Sie einen Ausdruck ein, der zutreffen muß (TRUE), wenn der Breakpoint ausgelöst werden soll. Der Ausdruck wird jedesmal ausgewertet, wenn das Programm zum Breakpoint kommt. Achten Sie auf eventuelle Seiteneffekte.

Enter expression to evaluate
[Geben Sie einen Ausdruck ein, der ausgewertet werden soll]
Geben Sie einen Ausdruck ein, dessen Ergebnis Sie ermitteln wollen. Das Ergebnis wird zusammen mit seinem Datentyp in einem eigenen Fenster dargestellt, das auf den nächsten Tastenanschlag wieder verschwindet.

Enter expression to watch
[Geben Sie den Ausdruck ein, der überwacht werden soll]
Geben Sie einen Variablennamen oder einen Ausdruck ein, dessen Wert Sie im *Watches*-Fenster überwachen wollen. Wenn Sie wollen, dann können Sie auch einen Ausdruck eingeben, der sich nicht auf eine Speicherstelle bezieht, wie zum Beispiel x * y + 4. Wenn das Dialogfenster mit einem Textausschnitt initialisiert wurde, kann man den Eintrag mit ↵ sofort akzeptieren oder verändern und etwas anderes eingeben.

Enter inspect start index, range
[Geben Sie den Index des ersten Elements und den Bereich ein]
Geben Sie den Index des ersten Array-Elements ein, das Sie betrachten wollen. Als zweites Argument müssen Sie die Anzahl der Elemente eingeben. Die beiden Zahlen müssen mit einem Leerzeichen oder einem Komma (,) voneinander getrennt werden.

Enter instruction to assemble
[Geben Sie den Befehl ein, der assembliert werden soll]
Geben Sie eine Assembleranweisung ein, die an der aktuellen Adresse des Code-Ausschnitts eingefügt werden soll. In Anhang B »Inline-Assemblerbefehle« finden Sie eine Aufstellung mit allen zulässigen Assemblerbefehlen. In Kapitel 12 »Debuggen auf Assemblerebene« wird der Einsatz von Assemblerbefehlen ausführlich behandelt.

Enter log file name
[Geben Sie den Namen der Protokolldatei an]
Geben Sie den Namen der Datei an, in die das Protokoll des *Log*-Fensters geschrieben werden soll. Bis zu dem Zeitpunkt, da Sie den Befehl *Close log file* aufrufen, werden alle Zeilen, die im *Log*-Fenster ausgegeben werden, auch in die Datei geschrieben. Die Voreinstellung für den Dateinamen ist die Dateierweiterung ».LOG« mit dem Namen der gerade bearbeiteten Datei. Mit ⊟ können Sie diese Vorgabe akzeptieren oder Sie geben stattdessen einen neuen Namen ein.

Enter memory address
[Geben Sie die Speicheradresse ein]
Geben Sie eine einzelne Speicherplatzadresse ein. Sie können einen Symbolnamen oder einen vollständigen Ausdruck verwenden.

Enter memory address, count
[Geben Sie die Adresse und Länge eines Speicherbereichs ein]
Geben Sie eine Speicherplatzadresse ein. Optional kann nach einem Komma die Anzahl der Elemente angegeben werden. Sie können einen Symbolnamen oder einen vollständigen Ausdruck verwenden.

Enter name of configuration file
[Geben Sie den Namen der Konfigurationsdatei ein]
Geben Sie den Namen einer Konfigurationsdatei ein, die gelesen oder geschrieben werden soll. Wenn Sie eine Konfigurationsdatei einlesen wollen, können Sie Jokerzeichen verwenden, um eine Liste von passenden Dateien zu erhalten.

Enter name of file to view
[Geben Sie den Namen der Datei ein, die dargestellt werden soll]
Sie können entweder ein Jokerzeichen, wie es von DOS verwendet wird, eingeben, um eine Liste mit Dateien zu erhalten, oder Sie geben den Dateinamen direkt ein.

Enter new bytes
[Geben Sie die neuen Bytes ein]
Geben Sie eine Liste von Bytes ein, um die Bytes zu ersetzen, die an der Stelle der Datei stehen, an der sich der Cursor befindet. In Kapitel 9 »Ausdrücke« finden Sie eine vollständige Beschreibung von Byte-Folgen.

Enter new coprocessor register value
[Geben Sie den neuen Wert für das Register des Koprozessors ein]
Geben Sie einen neuen Wert für das Register des Arithmetikprozessors ein, das momentan hervorgehoben ist. Sie können einen vollständigen Ausdruck eingeben, um den neuen Wert zu erzeugen. Der Ausdruck wird auf das richtige Gleitkommaformat konvertiert, bevor er in das Register geladen wird.

Enter new data bytes
[Geben Sie die neuen Daten-Bytes ein]
Geben Sie eine Liste von Bytes ein, um die Bytes zu ersetzen, die an der Stelle des Segments stehen, das vom Cursor gekennzeichnet ist. In Kapitel 9 »Ausdrücke« finden Sie eine vollständige Beschreibung der Byte-Folgen.

Enter new directory
[Geben Sie ein neues Verzeichnis ein]
Geben Sie die neue Laufwerksbezeichnung und/oder den Namen des Verzeichnisses ein, das Sie als aktuelles Laufwerk oder Verzeichnis verwenden möchten.

Enter new file offset
[Geben Sie die neue Position in der Datei ein]
Sie lassen den Inhalt einer Datei als Hexadezimal-Listing anzeigen. Geben Sie, bezogen auf den Anfang der Datei, einen neuen Wert für die Position in der Datei an, ab der die Daten in hexadezimaler Form anzuzeigen sind.

Enter new line number
[Geben Sie die Nummer der neuen Zeile ein]
Geben Sie die Nummer der Zeile des aktuellen Moduls ein, die Sie sehen wollen. Wenn Sie eine Zeilennummer eingeben, die hinter dem Ende der Datei liegt, kommen Sie in die letzte Zeile. Die Numerierung der Zeilen beginnt mit Eins. Die aktuelle Zeile wird durch den Cursor im *Module*-Fenster angezeigt.

Enter new relocation segment value
[Geben Sie einen neuen Wert für das Basissegment ein]
Geben Sie einen Ausdruck in der aktuellen Programmiersprache ein. Dieser Wert legt die Adresse des Basissegments der Symboltabelle fest, die mit dem Befehl *File / Symbol Load* geladen wurde. Der eingegebene Ausdruck sollte dem Segment entsprechen, an dem der Programmcode der entsprechenden Symboltabelle beginnt.

Enter new value
[Geben Sie einen neuen Wert ein]
Geben Sie für das momentan hervorgehobene CPU-Register einen neuen Wert ein. Sie können einen vollständigen Ausdruck eingeben, um den neuen Wert festzulegen.

Enter port number
[Geben Sie die Portadresse ein]
Geben Sie die Adresse des Eingabeports an, den Sie lesen wollen. Als Portadresse sind Werte zwischen 0 und 65535 zulässig.

Enter port number, value to output
[Geben Sie die Portadresse und den Wert ein, der ausgegeben werden soll]
Geben Sie die Adresse des Ausgabeports und den Wert an, der ausgegeben werden soll. Die beiden Ausdrücke müssen durch ein Komma voneinander getrennt werden. Als Portadresse sind Werte zwischen 0 und 65535 zulässig.

Enter program name to load
[Geben Sie den Namen des Programms ein, das geladen werden soll]
Geben Sie den Namen des Programms ein, das Sie debuggen wollen. Sie können DOS-Jokerzeichen benützen, um eine Auswahlliste von Dateien zu erhalten, oder den Dateinamen direkt eingeben, um das Programm zu laden. Wenn Sie den Dateinamen ohne Erweiterung eingeben, wird automatisch die Erweiterung ».EXE« angehängt.

Enter read file name
[Geben Sie den Namen des Programms ein, das geladen werden soll]
Geben Sie den Namen der Datei ein, die Sie einlesen wollen. Wenn Sie Jokerzeichen benützen oder die Voreinstellung »***.***« verwenden, erhalten Sie eine Auswahlliste mit den verfügbaren Dateien.

Enter search bytes
[Geben Sie die gesuchten Bytes ein]
Geben Sie eine Byte-Folge ein, die ab der Speicherstelle, in der der Cursor steht, gesucht werden soll. In Kapitel 9 finden Sie eine vollständige Beschreibung der Byte-Listen.

Enter search instruction or bytes
[Geben Sie den Befehl oder die Bytes ein, die gesucht werden sollen]
Geben Sie eine Anweisung ein, die so aussieht, als würden Sie auf den Befehl *Assemble* im lokalen Menü antworten, oder geben Sie eine Byte-Folge wie bei einem *Search*-Befehl in einem Datenausschnitt ein.

Enter search string
[Geben Sie den gesuchten String ein]
Geben Sie den String ein, der gesucht werden soll. Sie können einfache Jokerzeichen verwenden, um ähnliche Strings zu suchen. Zum Beispiel kann das Sternchen (*****) für null oder mehrere beliebige Zeichen und das Fragezeichen (?) für ein beliebiges Zeichen verwendet werden.

Enter source address, destination, count
[Geben Sie Quell- und Zieladresse sowie die Anzahl ein]
Geben Sie die Adresse des Blocks ein, den Sie kopieren möchten, die Adresse, wohin Sie ihn kopieren wollen, und die Anzahl der Byte, die Sie kopieren wollen. Trennen Sie die drei Ausdrücke durch Kommata voneinander.

Enter source directory path
[Geben Sie die Verzeichnispfade für Quelldateien ein]
Geben Sie eine Liste von Verzeichnissen ein, die durch Leerzeichen oder Strichpunkte (;) getrennt sind. Diese Verzeichnisse werden in der Reihenfolge, in der sie in dieser Liste erscheinen, nach den Quelldateien durchsucht.

Enter symbol table name
[Geben Sie den Namen der Symboltabelle ein]
Geben Sie den Namen der Symboldatei ein, die geladen werden soll. Üblicherweise haben diese Dateien die Erweiterung ».TDS«. Die Dateierweiterung muß explizit angegeben werden.

Enter tab column spacing
[Geben Sie die Schrittweite der Tabulatoren ein]
Geben Sie eine Zahl zwischen 1 und 32 ein, die festlegt, wie weit die Tabulatoren voneinander entfernt sein sollen, wenn Turbo-Debugger-Dateien in den Fenstern *File* oder *Module* anzeigt.

Enter variable to inspect
[Geben Sie die Variable ein, die untersucht werden soll]
Geben Sie den Namen der Variablen oder des Ausdrucks ein, dessen Inhalt Sie untersuchen wollen. Falls das Dialogfenster mit einem Wert aus einem Fensterausschnitt initialisiert wird, können Sie den Eintrag sofort mit der Eingabetaste ⏎ bestätigen oder nach Belieben ändern.

Enter write file name
[Geben Sie den Namen der Datei ein, in die die Informationen geschrieben werden sollen]
Geben Sie den Namen der Datei ein, in die Sie den Speicherblock schreiben wollen.

Overwrite __ ?
[__ überschreiben ?]
Sie haben den Namen einer Datei angegeben, die bereits existiert. Sie können die Datei überschreiben, und damit den bisherigen Inhalt ersetzen, oder den Befehl abbrechen und die bisherige Datei beibehalten.

Overwrite existing macro on selected key?
[Soll das vorhandene Makro für die ausgewählte Tastenkombination überschrieben werden?]
Sie haben eine Tastenkombination gedrückt, um ein Makro aufzunehmen, und dieser Tastenkombination ist schon ein Makro zugeordnet. Wenn Sie das bestehende Makro überschreiben wollen, geben Sie [Y] ein, sonst drücken Sie [N], um den Befehl rückgängig zu machen.

Pick a method name
[Wählen Sie eine Methode aus]
Sie haben den Namen einer Routine angegeben, der sich auf mehr als ein Objekt bezieht. Wählen Sie die gewünschte Routine in der Liste aus.

Pick a module
[Wählen Sie ein Modul aus]
Wählen Sie den Namen eines Moduls aus, das im *Module*-Fenster angezeigt werden soll. Sie erhalten eine Liste aller Module in Ihrem Programm. Wenn Sie eine Datei anschauen möchten, die kein Programmodul ist, verwenden Sie den Menübefehl *View / File*.

Pick a source file
[Wählen Sie eine Quelldatei aus]
Wählen Sie eine Quelldatei aus der Liste auf dem Bildschirm aus. Es werden nur die Quelldateien gezeigt, die zum aktuellen Modul gehören.

Pick a symbol
[Wählen Sie ein Symbol aus]
Wählen Sie ein Symbol aus der dargestellten Liste an Symbolen aus. Sie können die Auswahl durch die Eingabe der Anfangsbuchstaben des Symbols beschleunigen. Der Auswahlbalken springt zum jeweils ersten Symbol, das mit den bis dahin eingegebenen Buchstaben übereinstimmt.

Pick a window
[Wählen Sie ein Fenster aus]
Wählen Sie ein Fenster aus der Liste mit den Überschriften der aktiven Fenster aus.

Press key to assign macro to
[Drücken Sie die Taste, der das Makro zugeordnet werden soll]
Drücken Sie die Taste, der Sie das Makro zuordnen wollen. Dann geben Sie die Befehlsfolge ein, die Sie der gewählten Makrotaste zuordnen wollen. Die Befehlsfolge wird tatsächlich ausgeführt, während Sie sie eingeben. Um die Makroaufnahme zu beenden, drücken Sie noch einmal die Taste, der Sie das Makro zugeordnet haben. Dieses Makro wird mit den anderen Tastaturmakros auf dem Laufwerk gespeichert.

Press key to delete macro from
[Drücken Sie die Taste, deren Makro gelöscht werden soll]
Drücken Sie die Taste des Makros, das Sie löschen möchten. Die Taste bekommt dann wieder die Funktion, die Sie vor der Zuordnung des Makros hatte.

Program already terminated; reload?
[Programm bereits beendet; neu laden?]
Sie haben versucht, Ihr Programm laufen zu lassen oder einen Befehl auszuführen, nachdem das Programm bereits beendet wurde. Wenn Sie Y eingeben, wird Ihr Programm neu gestartet, wenn Sie mit N antworten, wird Ihr Programm nicht neu gestartet und die Befehle zum Ablaufen des Programms werden nicht ausgeführt.

Reload program so arguments take effect?
[Programm neu laden, damit die Argumente übernommen werden?]
Sie haben gerade die Kommandozeilenargumente für das Programm geändert, das Sie debuggen. Wenn Sie Y eingeben, wird Ihr Programm neu geladen und an den Programmanfang gesetzt. Sie müssen das normalerweise dann machen, nachdem Sie die Argumente geändert haben. Viele Programme, die mit Programmiersprachen von Borland erstellt wurden, überprüfen die Argumente nur einmal: wenn das Programm geladen wird. Alle nachfolgenden Änderungen der Programmargumente werden nicht beachtet, bis das Programm neu gestartet wird.

Fehlermeldungen

Turbo Debugger verwendet Fehlermeldungen, um Ihnen Informationen zu unerwarteten Vorgängen zu geben. Manchmal bedeutet das, daß ein Befehl nicht ausgeführt werden konnte.

Fehlermeldungen werden normalerweise von einem Alarmton begleitet. Sie können den Alarmton mit Hilfe des Installationsprogramms TDWINST abschalten.

Schwerwiegende Fehlermeldungen

Bei allen schwerwiegenden Fehlern verabschiedet sich Turbo Debugger und kehrt sofort zu DOS zurück. Manche dieser Fehler treten beim Aufruf des Debuggers auf. Andere passieren dann, wenn etwas Irreparables geschieht, während man mit Turbo Debugger arbeitet. In beiden Fällen haben Sie keine andere Möglichkeit, als den Debugger nach der Lösung des Problems erneut zu starten.

Bad configuration file
[Fehlerhafte Konfigurationsdatei]
Die Konfigurationsdatei ist entweder zerstört oder keine Konfigurationsdatei für Turbo Debugger.

Invalid switch: ___
[Unzulässige Option: ___]
Sie haben beim Aufruf des Debuggers eine unzulässige Option angegeben. Anhang A »Kommandozeilenoptionen« enthält eine Aufstellung mit allen Kommandozeilenoptionen. In Kapitel 4 »Der Aufruf von Turbo Debugger« werden diese Optionen eingehend erläutert.

Not enough memory
[Zuwenig Speicherplatz frei]
Turbo Debugger hat nicht mehr genug Speicherplatz, um Ihren Befehl auszuführen.

Old configuration file
[Veraltete Konfigurationsdatei]
Sie haben versucht, Turbo Debugger mit einer Konfigurationsdatei einer früheren Version zu starten. Sie müssen neue Konfigurationsdateien für die verwendete Version von Turbo Debugger anlegen.

Unsupported video adapter
[Video-Adapter wird nicht unterstützt]
Turbo Debugger kann nicht feststellen, welchen Bildschirmadapter Sie verwenden. Es werden nur Adapter des Typs EGA, VGA, Hercules und Super-VGA unterstützt.

Andere Fehlermeldungen

')' expected
[')' erwartet]
Bei der Auswertung eines Ausdrucks wurde festgestellt, daß eine abschließende Klammer fehlt. Dies passiert, wenn ein korrekter Ausdruck mit einer linken Klammer anfängt, aber nicht mit einer passenden rechten Klammer abgeschlossen wird. Zum Beispiel sollte

```
3*(7+4
```

wie folgt lauten:

```
3*(7+4)
```

':'expected
[':' erwartet]
Bei der Auswertung eines C-Ausdrucks wurde ein Fragezeichen (?) gefunden, mit dem die beiden ersten Ausdrücke des dreiwertigen Operators ?: getrennt

sind. Es wurde jedoch kein entsprechender Doppelpunkt (:) gefunden, der den zweiten und dritten Ausdruck voneinander getrennt hätte. Zum Beispiel sollte

```
x < 0 ? 4 6
```

wie folgt formuliert werden:

```
x < 0 ? 4 : 6
```

']' expected
[']' erwartet]
Bei der Auswertung eines Ausdrucks wurde festgestellt, daß eine abschließende eckige Klammer fehlt. Dies passiert, wenn ein Array-Index mit einer linken eckigen Klammer anfängt, aber nicht mit einer passenden rechten eckigen Klammer abgeschlossen wird. Zum Beispiel sollte

```
Tabelle [4
```

wohl

```
Tabelle [4]
```

heißen. Dieser Fehler kann auch dann auftreten, wenn ein Assemblerbefehl mittels des integrierten Assemblers eingegeben wurde. In diesem Fall wurde eine linke eckige Klammer bei einem Speicherzugriff mit einem Basis- oder einem Indexregister angetroffen, ohne daß der Ausdruck mit der rechten eckigen Klammer abgeschlossen worden wäre. Zum Beispiel sollte

```
mov     ax,4[si
```

als

```
mov     ax,4[si]
```

geschrieben werden.

Already logging to a file
[Protokoll erfolgt bereits auf Datei]
Sie haben den Befehl *Open log file* angegeben, obwohl derselbe Befehl bereits eingegeben wurde, ohne daß zwischendurch der Befehl *Close log file* erfolgt wäre. Wenn Sie eine andere Protokolldatei verwenden wollen, müssen Sie die aktuelle Protokolldatei vorher mit dem Befehl *Close log file* schließen.

Ambiguous symbol name
[Mehrdeutige Symbolbezeichnung]
Sie haben einen Symbolnamen in einem Ausdruck eingegeben, der sich innerhalb eines Programms in C++ oder objektorientierten Pascal nicht eindeutig zuordnen läßt, und darüber hinaus kein Symbol aus der angebotenen Liste ausgewählt. Sie müssen das entsprechende Symbol aus der dargestellten Liste auswählen, bevor der Ausdruck ausgewertet werden kann.

Assignment out of range
[Zuweisung außerhalb des Wertebereichs]
Sie haben in Pascal versucht, einer Variablen einen Wert zuzuweisen, der außerhalb ihres Wertebereichs liegt.

Bad configuration file name
[Falscher Name der Konfigurationsdatei]
Die hinter der Kommandozeilenoption -c angegebene Konfigurationsdatei existiert nicht.

Cannot be changed
[Kann nicht verändert werden]
Sie haben versucht, ein Symbol zu ändern, das nicht verändert werden kann. Die einzigen Symbole, die direkt verändert werden können, sind skalare Daten (Byte, Integer, LongInt usw.) und Zeiger. Wenn Sie einen Record oder ein Array verändern wollen, müssen Sie die Elemente einzeln nacheinander verändern.

Can't have more than one segment override.
[Kann nicht mehr als eine Segmentvorgabe verwenden]
Sie wollten einen Befehl assemblieren, in dem beide Operanden eine Segmentvorgabe hatten. Nur einer der beiden Operanden darf jedoch eine Segmentvorgabe besitzen. Zum Beispiel sollte

```
mov    es:[bx],ds:ax
```

als

```
mov    es:[bx],ax
```

geschrieben werden.

Can't set a breakpoint at this address
[Breakpoint kann nicht an dieser Adresse gesetzt werden]
Sie wollten einen Breakpoint im ROM, in einem nicht vorhandenen Speicherbereich oder im Segment 0 setzen. Die einzige Möglichkeit, um die Ausführung eines Programms im ROM nachzuvollziehen, ist der Befehl *Run/Trace*, bei dem man Befehl für Befehl beobachten kann.

Can't use same register twice
[Dasselbe Register darf nicht zweimal verwendet werden]
Sie wollten einen Befehl assemblieren, in dem ein Basis- oder Indexregister zweimal im selben Speicheroperanden verwendet wird. Sie können ein Register nur einmal in einem Operanden verwenden. Zum Beispiel ist

```
mov    ax,[bx+bx]
```

nicht erlaubt, während folgende Formulierung durchaus verwendet werden kann:

```
mov    ax,[bx+si]
```

Cannot access an inactive scope
[Auf einen nicht aktivierten Gültigkeitsbereich kann nicht zugegriffen werden]
Sie haben einen Ausdruck eingegeben oder auf eine Variable in einem *Module*-Fenster zugegriffen, die nicht im Bereich der aktiven Funktion liegt. Variablen in einer nicht aktivierten Funktion haben keinen definierten Wert, deshalb können Sie sie nicht in Ausdrücken verwenden oder deren Werte anschauen.

Constructors and destructors cannot be called
[Konstruktoren und Destruktoren können nicht aufgerufen werden]
Diese Fehlermeldung erscheint nur dann, wenn Sie ein Programm debuggen, in dem Objekte verwendet werden. Sie haben wahrscheinlich versucht, eine Objektmethode oder Elementfunktion auszuwerten, das entweder ein Konstruktor oder ein Destruktor ist. Dies ist nicht zulässig.

Ctrl-Alt-SysReq interrupt. System crash possible. Continue?
[ctrl-Alt-SysReq gedrückt. Systemabsturz möglich. Weitermachen?]
Sie versuchen entweder TDW zu verlassen oder das zu untersuchende Programm neu zu laden, während das Programm angehalten von Ihnen mittels [Ctrl][Alt][SysReq] angehalten wurde. Da zu dem Zeitpunkt, als Sie die Programmausführung anhielten, Code des Windows-Kernel abgearbeitet wurde, führt das Verlassen von TDW oder das erneute Laden Ihres Programms zu nicht vorhersehbaren Ergebnissen. Wahrscheinlich wird Ihr Computer hängen, und Sie sind gezwungen, neu zu booten.

Setzen Sie wenn möglich einen Breakpoint in Ihrem Programm, bei dessen Erreichen TDW die Kontrolle wieder übernimmt. An diesem Punkt können Sie gefahrlos TDW beenden oder Ihr Programm erneut laden.

Destination too far away
[Zieladresse zu weit entfernt]
Sie wollten einen bedingten Sprungbefehl assemblieren, bei dem die Zieladresse zu weit von der aktuellen Position entfernt ist. Die Zieladresse eines bedingten Sprungbefehls muß innerhalb eines Bereichs von -128 und +127 Bytes relativ zur aktuellen Position entfernt liegen.

Divide by zero
[Division durch Null]
Sie haben einen Ausdruck mit einem Divisions- (/, div) oder Modulo-Operator (mod, %) eingegeben, dessen Ausdruck auf der rechten Seite den Wert Null ergab. Da die Operatoren in diesem Fall kein definiertes Ergebnis haben, wurde eine Fehlermeldung ausgegeben.

Error loading program
[Fehler beim Laden des Programms]
DOS war nicht in der Lage, das Programm zu starten, das Sie angegeben haben. Das könnte entweder heißen, daß die angegebene Datei keine gültige ».EXE«-Datei ist, oder daß die ».EXE«-Datei zerstört ist.

Error opening file ___
[Fehler beim Öffnen der Datei ___]
Turbo Debugger war nicht in der Lage, die Datei zu öffnen, die Sie im Datei-Fenster anschauen wollen.

Error opening log file ___
[Fehler beim Öffnen der Log-Datei ___]
Die Datei, die Sie im Befehl *Log to file* angegeben haben, konnte nicht geöffnet werden. Entweder ist nicht mehr genug Speicherplatz vorhanden, um die Datei zu öffnen, oder das angegebene Laufwerk, Verzeichnis oder der Name der Datei ist nicht zulässig. Machen Sie entweder etwas Platz, indem Sie einige Dateien auf dem Laufwerk löschen oder geben Sie einen zulässigen Pfad für die Datei an.

Error reading block into memory
[Fehler beim Einlesen eines Speicherblocks]
Der von Ihnen angegebene Block konnte nicht von der Diskette in den Speicher geladen werden. Sie haben wahrscheinlich mehr Bytes einlesen wollen, als in der Datei vorhanden sind.

Error recording keystroke macros
[Fehler bei der Aufnahme eines Tastaturmakros]
Während des Schreibens des aufgenommenen Tastaturmakros in die Konfigurationsdatei ist ein Fehler aufgetreten. Das Makro ist wahrscheinlich nicht gespeichert.

Error saving configuration
[Fehler beim Speichern der Konfigurationsdatei]
Turbo Debugger konnte Ihre Konfiguration nicht auf das Laufwerk schreiben. Vergewissern Sie sich, daß auf Ihrem Laufwerk genug Speicherplatz frei ist.

Error swapping in user program, press key to reload
[Fehler beim Auslagern des Programms, drücken Sie eine Taste, um das Programm neu zu laden]
Nachdem Turbo Debugger Ihr Programm auf ein Laufwerk übertragen hat, um ein anderes Programm auszuführen, kann er Ihr Programm nicht neu starten. Das bedeutet höchstwahrscheinlich, daß Sie aus Versehen die Datei SWAP.$$$ gelöscht haben, in die Ihr Programm übertragen worden ist. Das Einzige, was der Debugger tun kann, ist, Ihr Programm neu zu starten und zwar genau so, als ob Sie den Menübefehl *File / Open* eingegeben hätten.

Error writing block to disk
[Fehler beim Schreiben eines Speicherblocks auf das Laufwerk]
Der angegebene Block konnte nicht in die angegebene Datei geschrieben werden. Wahrscheinlich überschreitet der Block den noch verfügbaren Speicherplatz auf dem Laufwerk.

Error writing log file
[Fehler beim Schreiben der Protokolldatei]
Turbo Debugger konnte die Ausgaben im Fenster *Log* nicht in die angegebene Datei schreiben, weil es zu einem Fehler kam. Möglicherweise ist nicht genug Platz auf dem Laufwerk frei.

Error writing to file
[Fehler beim Schreiben in eine Datei]
Turbo Debugger konnte Ihre Änderungen nicht in die Datei zurückschreiben. Die Datei war möglicherweise schreibgeschützt oder während des Schreibvorgangs ist ein Hardware-Fehler aufgetreten.

Expression accesses more than one scope
[Ausdruck greift auf mehr als einen Gültigkeitsbereich zu]
In Verbindung mit einem Breakpoint haben Sie einen Ausdruck eingegeben, der sich auf Variablen aus zu vielen Gültigkeitsbereichen bezieht. In Pascal kann man lokale Variablen und Parameter sowie globale und lokale Variablen aus einem umschließenden Unterprogramm angeben (wenn der Breakpoint in einer verschachtelten Prozedur oder Funktion liegt).

Expression too complex
[Ausdruck zu komplex]
Der Ausdruck, den Sie eingegeben haben, ist zu komplex. Sie müssen einen Ausdruck eingeben, der weniger Operatoren und Operanden hat. Sie können bis zu 64 Operatoren und Operanden in einem Ausdruck verwenden. Operanden sind zum Beispiel Konstanten und Namen von Variablen. Operatoren sind zum Beispiel das Pluszeichen (+), der Zuweisungsoperator (= oder :=), eine Elementauswahl in Strukturen – und eine Mengenzuordnung (in).

Expression with side effects not permitted
[Ausdruck mit Seiteneffekten nicht zulässig]
Sie haben einen Ausdruck eingegeben, der bei der Auswertung eine Speicherstelle verändert. Sie können diese Art von Ausdrücken nicht angeben, wenn Turbo Debugger einen Ausdruck möglicherweise wiederholt auswerten muß, wie zum Beispiel in einem *Inspector*- oder *Watches*-Fenster.

Extra input after expression
[Zusätzliche Zeichen hinter dem Ausdruck]
Sie haben einen Ausdruck eingegeben, der zwar gültig war, aber hinter dem Ausdruck stehen weitere Angaben. Das kann heißen, daß Sie einen Operator in Ihrem Ausdruck vergessen haben. Zum Beispiel kann es sein, daß

```
3 * 4 + 5 2
```

folgendes

```
3 * 4 + 5 / 2
```

heißen sollte. Oder

```
add    ax,4 5
```

sollte

```
add    ax,45
```

heißen.

Help file ___ not found
[Hilfedatei ___ nicht gefunden]
Sie haben nach Hilfestellungen gefragt, aber die Datei mit den Hilfebildschirmen konnte nicht gefunden werden. Stellen Sie sicher, daß sich die Hilfedatei im selben Verzeichnis wie das Debuggerprogramm befindet.

Illegal procedure or function call
[Unzulässiger Prozedur- oder Funktionsaufruf]
Sie wollten eine Funktion zu einem Zeitpunkt auswerten, da dies nicht möglich war. Dies kann in einem der drei folgenden Fälle passieren:

- Sie wollten eine Funktion aufrufen, die in einem Pascal-Overlay steht.
- Sie wollten eine Funktion aufrufen, während Sie sich in einem Pascal-Overlay aufhielten.
- Sie wollten eine Methode in einem objektorientierten Pascal-Programm aufrufen, die vom intelligenten Linker von Turbo Pascal entfernt worden war.

Immediate Operand out of range
[Konstanter Operand zu groß]
Sie haben eine Anweisung eingegeben, die einen Operanden in Byte-Größe mit einer Konstanten kombiniert, die zu groß für ein Byte ist. Zum Beispiel sollte

```
add    BYTE PTR [bx],300
```

vielleicht

```
add    WORD PTR [bx],300
```

heißen.

Initialization not complete
[Initialisierung nicht vollständig]
Sie haben versucht auf eine Variable in Ihrem Programm zuzugreifen, bevor das Datensegment vom Initialisierungscode des Compilers richtig geladen wurde. Sie müssen den Initialisierungscode des Compilers bis zum Beginn Ihres Quellcodes ausführen, bevor Sie auf die meisten Programmvariablen zugreifen können.

Invalid argument list
[Unzulässige Argumentliste]
Der Ausdruck, den Sie eingegeben haben, enthält einen Prozedur- oder Funktionsaufruf, der keine korrekt gebildete Argumentliste besitzt. Eine Argu-

mentliste beginnt mit einer linken Klammer, hat null oder mehr Argumentausdrücke, die durch ein Komma getrennt sind, und schließt mit einer rechten Klammer ab. Bitte beachten Sie, daß Turbo Debugger leere Klammern voraussetzt, um eine Pascal-Funktion oder -Prozedur ohne Parameter aufzurufen. Zum Beispiel:

```
funk(1,2 3)
```

sollte heißen

```
funk(1,2,3)
```

oder auch

```
funk()
```

Invalid character constant
[Unzulässige Zeichenkonstante]
Der eingegebene Ausdruck enthält eine falsche Angabe einer Zeichenkonstanten. Eine Zeichenkonstante besteht aus einem einfachen Anführungszeichen (’), hinter dem ein einziges Zeichen steht und das mit einem weiteren einfachen Anführungszeichen abgeschlossen ist. Zum Beispiel:

```
'A = 'a'
```

sollte heißen

```
'A'= 'a'
```

Invalid far address
[Ungültige FAR-Adresse]
Beim Assemblieren eines Befehls haben Sie eine falsche FAR-Adresse als Zeile für einen JMP- oder CALL-Befehl eingegeben. Eine FAR-Adresse besteht aus einem Paar von Hexadezimalzahlen, die durch einen Doppelpunkt getrennt sind. Zum Beispiel:

```
JMP    1234:XYZ
```

sollte heißen

```
JMP    1234:1000
```

Invalid format string
[Ungültiger Formatstring]
Sie haben eine Formatierungsangabe hinter einem Ausdruck eingegeben. Der Formatierungsstring ist jedoch ungültig. In Kapitel 9 »Ausdrücke« finden Sie eine Beschreibung der Formatierungsangaben.

Invalid function parameters
[Ungültige Funktionsparameter]
Sie haben in einem Ausdruck versucht, eine Funktion aufzurufen, haben aber nicht die richtigen Parameter zum Funktionsaufruf eingegeben.

Invalid instruction
[Unzulässiger Befehl]
Sie haben einen Befehl zum Assemblieren eingegeben, der eine gültige Befehlskurzform hatte, aber der von Ihnen angegebene Operand ist nicht zulässig. Das passiert meistens beim Versuch, einen POP CS-Befehl zu assemblieren.

Invalid instruction mnemonic
[Unzulässiger Befehl]
Bei der Eingabe eines Befehls, der assembliert werden soll, haben Sie die Befehlskurzform vergessen. Ein Befehl besteht aus einer Befehlskurzform, gefolgt von optionalen Argumenten. Zum Beispiel:

```
AX,123
```

sollte heißen

```
MOV     ax,123
```

Invalid operand separator
[Ungültiges Trennzeichen zwischen den Operanden]
Sie haben einen Befehl zum Assemblieren eingegeben, haben aber die Operanden nicht durch Kommas getrennt. Wenn ein Maschinensprachebefehl mehr als einen Operanden hat, müssen Sie immer ein Komma zwischen den Operanden angeben. Zum Beispiel:

```
ADD     ax 12
```

sollte heißen

```
ADD     ax,12
```

Invalid operand(s)
[Unzulässig(e/r) Operand(en)]
Der Befehl, den Sie assemblieren wollen, enthält einen oder mehrere Operanden, die nicht zugelassen sind. Zum Beispiel kann ein MOV-Befehl nicht zwei Speicheroperanden enthalten und einige Befehle arbeiten nur mit Word-Operanden. Zum Beispiel:

```
POP     al
```

sollte heißen

```
POP     ax
```

Invalid operator/data combination
[Ungültige Kombination aus Daten und Operatoren]
Sie haben einen Ausdruck eingegeben, in dem ein Operator einen Operanden hat, mit dem die angegebene Operation nicht durchgeführt werden kann. Das ist zum Beispiel der Fall, wenn Sie eine Konstante mit der Adresse einer Funktion in Ihrem Programm multiplizieren wollen.

Invalid pass count entered
[Ungültige Vorgabe für Durchlaufzähler]
Sie haben einen Durchlaufzähler für einen Breakpoint eingegeben, der nicht zwischen 1 und 65535 liegt. Sie dürfen für den Durchlaufzähler nicht 0 eingeben. Während Ihr Programm läuft, bedeutet ein Durchlaufzähler von 1, daß der Breakpoint bereits beim ersten Mal ausgelöst wird.

Invalid register
[Ungültiges Register]
Sie haben ein ungültiges Gleitkommaregister als Teil eines Befehls eingegeben, der assembliert werden sollte. Ein Gleitkommaregister besteht aus den Buchstaben ST, hinter denen optional eine Zahl von 0 bis 7 in Klammern stehen kann, wie zum Beispiel ST(4).

Invalid register combination in address expression
[Unzulässige Registerkombination im Adreßausdruck]
Bei der Eingabe eines Befehls, der assembliert werden sollte, haben Sie einen Operanden angegeben, der keine der zugelassenen Kombinationen von Basis- und Indexregister enthielt. Ein Adreßausdruck kann ein Basisregister, ein Indexregister oder eines von beiden enthalten. Die Basisregister sind BX und BP, die Indexregister sind SI und DI. Hier die gültigen Adreßregisterkombinationen:

```
BX    BX+SI
BP   BP+SI
DI   BX+DI
SI   BP+DI
```

Invalid register in address expression
[Unzulässiges Register im Adreßausdruck]
Sie haben einen Befehl zum Assemblieren eingegeben, in dem Sie ein unzulässiges Register in den Klammern ([ ]) eines Adreßausdrucks angegeben haben. In Adreßausdrücken darf man nur die Register BX, BP, SI und DI verwenden.

Invalid symbol in operand
[Unzulässiges Symbol im Operanden]
Beim Assemblieren eines Befehls, haben Sie einen Operanden mit einem Zeichen eingeleitet, das nicht am Anfang eines Operanden stehen darf, wie zum Beispiel der Doppelpunkt (:).

Invalid typecast
[Unzulässige Syntax bei Datentypvorgabe]
Sie haben einen Ausdruck eingegeben, der eine explizite Typvorgabe enthält, die nicht korrekt ist. Eine korrekte explizite Typvorgabe beginnt in Pascal mit einem bekannten Datentyp, enthält dann eine linke Klammer, dann einen Ausdruck und endet mit einer rechten Klammer. Zum Beispiel

```
Longint(p)
```

oder

```
Word(p^)
```

Invalid value entered
[Ungültiger Wert eingegeben]
Als Sie nach einer Speicheradresse gefragt wurden, haben Sie eine Gleitkommazahl statt einer Integerzahl eingegeben.

Invalid window handle
[Ungültiges Fenster-Handle eingegeben]
Sie wollen die einem Fenster zugehenhden Botschaften protokollieren, haben aber eine Handle-Variable angegeben, der noch kein Handle zugeordnet ist. Lassen Sie Ihr Programm bis nach der Anweisung ausführen, die diese Zuordnung veranlaßt, und geben Sie den Variablennamen erneut ein.

Keyword not a symbol
[Reserviertes Wort darf kein Symbol sein] (Gilt nur für Assembler)
Der von Ihnen eingegebene Ausdruck enthält ein reserviertes Wort an einer Stelle, an der ein Variablenname erwartet wird. Sie dürfen reservierte Wörter nur in einer Datentypumwandlung verwenden.

Left side not a structure or union
[Auf der linken Seite steht keine Struktur oder Variante]
Sie haben einen Ausdruck eingegeben, in dem ein Pascal-Record-Feldselektor (.) steht. Vor diesem Symbol steht aber weder ein Recordname, noch ein Zeiger auf einen Record.

No coprocessor or emulator installed
[Kein Koprozessor oder Emulator installiert]
Sie haben das Fenster *Numeric processor* mit dem Befehl *View / Numeric processor* anlegen wollen, aber es ist kein Koprozessor installiert und der Software-Emulator wird ebenfalls nicht verwendet.

No help for this context
[Keine Hilfeinformationen in diesem Zusammenhang verfügbar]
Sie haben F1 gedrückt, um Hilfe zu erhalten, aber Turbo Debugger hat keinen entsprechenden Hilfebildschirm. Teilen Sie den Sachverhalt bitte schriftlich der Fa. Borland mit.

No modules with line number information
[Keine Module mit Zeilennummerinformationen]
Sie haben den Befehl *View / Module* verwendet, aber Turbo Debugger kann keine Module mit Debug-Informationen finden, um diese Module im Quelltext darzustellen. In der Regel erhalten Sie diese Meldung, wenn Sie ein

Programm ohne Symboltabelle debuggen. Sehen Sie dazu unter der Fehlermeldung »Program has no symbol table« nach.

No previous search expression
[Kein Suchbefehl vorausgegangen]
Sie wollten den Befehl *Next* in einem lokalen Menü eines Textausschnitts aufrufen, aber Sie haben vorher noch keinen *Search*-Befehl eingegeben. Sie können *Next* nur nach einem *Search*-Befehl verwenden.

No program loaded
[Kein Programm geladen]
Sie wollten einen Befehl eingeben, bei dem ein Programm geladen sein muß. Es kann zum Beispiel keiner der Befehle im *Run*-Menü verwendet werden, solange kein Programm geladen ist. Wählen Sie den Befehl *File / Open* aus, um ein Programm vor der Verwendung dieser Befehle zu laden.

No source file for module ___
[Keine Quelldatei für das Modul ___ verfügbar]
Für das Modul, das Sie anschauen wollen, ist kein Quelltext verfügbar. Wenn die Quelldatei nicht im aktuellen Verzeichnis ist, kann man den Befehl *Options / Path for source* verwenden, um anzugeben, wo das Verzeichnis mit den Quelldateien ist.

No type information for this symbol
[Keine Datentypinformation für dieses Symbol]
Sie haben einen Ausdruck eingegeben, der den Namen einer Variablen enthält, für die keine Debug-Informationen vorliegen. Das kann passieren, wenn die Variable in einem Modul vorkommt, das ohne die richtigen Debug-Informationen compiliert wurde. Sie können den Datentyp mit einer expliziten Datentypumwandlung im Ausdruck angeben.

Not a function name
[Kein Funktionsname]
Sie haben einen Ausdruck mit einem Funktionsaufruf eingegeben, aber die Bezeichnung vor der linken Klammer, mit der der Funktionsaufruf eingeleitet wird, ist kein Funkionsname. Jedesmal, wenn eine linke Klammer direkt hinter einem Namen steht, nimmt der Parser an, daß es sich um einen Funktionsaufruf handelt.

Not a memory referencing expression
[Ausdruck ergibt keine Speicherstelle]
Sie haben einen Ausdruck eingegeben, der sich nicht auf eine Speicherstelle bezieht. Es gibt viele Fälle, in denen ein Ausdruck eine Speicherstelle ergeben muß und nicht nur einen Wert liefern darf. Der Befehl *Data / Inspect* benötigt zum Beispiel einen Speicherbereich, und nicht nur ein Ergebnis im Ausdruck. Zum Beispiel bezieht sich

```
3 * 4 < (9 - 1)
```

nicht auf eine Speicherstelle, während dies bei

```
xarray[4]
```

der Fall ist.

Not an Object Pascal or C++ program
[Kein objektorientiertes Pascal- oder C++-Programm]
Ihr Programm ist nicht in objektorientiertem Pascal geschrieben. Es enthält daher auch keine Objekte. Aus diesem Grund kann der ausgewählte Befehl nicht durchgeführt werden.

Not a record, structure, or union member
[Kein Element eines Records, einer Struktur oder Variante]
Sie haben einen Ausdruck mit einem Pascal-Record-Selektor (.) angegeben. Hinter dem Symbol steht jedoch kein Element des Records.

Not enough memory for selected operation
[Nicht genug Speicherplatz für die ausgewählte Operation]
Sie haben einen Befehl eingegeben, bei dem ein Fenster angelegt werden muß, aber es ist nicht mehr genug Speicherplatz für das neue Fenster verfügbar. Sie müssen zuerst einige der anderen Fenster verkleinern oder entfernen, bevor Sie diesen Befehl erneut eingeben können.

Not enough memory to load program
[Zuwenig Speicherplatz, um das Programm zu laden]
Die Symboltabelle Ihres Programms konnte erfolgreich in den Speicher geladen werden, aber es ist nicht mehr genug Speicherplatz für Ihr Programm verfügbar.

Not enough memory to load symbol table
[Zuwenig Speicherplatz, um die Symboltabelle zu laden]
Es ist nicht genügend Speicherplatz verfügbar, um die Symboltabelle in den Speicher zu laden. Die Symboltabelle enthält Informationen, die Turbo Debugger zur Darstellung des Quelltextes und der Programmvariablen benötigt. Wenn Sie speicherresidente Programme geladen haben, sollten Sie diese entfernen und den Debugger noch einmal starten. Sie können auch versuchen, die Symboltabelle zu verkleinern, indem Sie beim Compilieren nur für diejenigen Module Debuggerinformationen erzeugen, die Sie tatsächlich debuggen wollen.

Wenn diese Meldung erscheint, bedeutet das, daß Ihr Programm selbst überhaupt noch nicht geladen wurde. Sie müssen also soviel Speicher freigeben, daß Symboltabelle und Programm in den Speicher passen.

Only one operand size allowed
[Nur eine Operandengröße zulässig]
Sie haben einen Befehl eingegeben, in dem mehr als ein Datentyp angegeben wurde. Sobald Sie die Größe des Operanden bestimmt haben, dürfen Sie sie nicht mehr verändern. Statt

```
mov    WORD PTR BYTE PTR [bx],1
```

hätten Sie besser

```
mov    BYTE PTR [bx],1
```

angeben sollen.

Operand must be memory location
[Operand muß eine Speicherstelle enthalten]
Sie haben einen Ausdruck eingegeben, in dem ein Teilausdruck eine Speicherstelle angeben sollte, was aber nicht der Fall war. Zum Beispiel muß man bei den Zuweisungsoperatoren (=, +=, usw.) und den Inkrement- oder Dekrement-Operatoren (++ und —) einen Speicheroperanden auf der linken Seite angeben.

Operand size unknown
[Operandengröße unbekannt]
Sie wollten einen Befehl assemblieren, haben aber die Größe eines Operanden nicht angegeben. Einige Befehle, die mit Bytes oder Worten arbeiten, benötigen eine genaue Angabe der Operandengröße, wenn diese aus den Operanden nicht ersichtlich ist. Statt

```
add    [bx],1
```

hätte man

```
add    BYTE PTR [bx],1
```

verwenden müssen.

Path not found
[Pfad nicht gefunden]
Sie haben ein Laufwerk mit einem Verzeichnis kombiniert, das nicht existiert. Überprüfen Sie, ob Sie das Laufwerk und das Verzeichnis korrekt eingegeben haben.

Path or file not found
[Pfad oder Datei nicht gefunden]
Sie haben einen Dateinamen oder einen Pfad für eine Datei eingegeben, die entweder nicht existiert oder deren Namen falsch eingegeben wurde. Wenn Sie nicht wissen, wie der exakte Name der Datei lautet, die Sie laden wollen, können Sie den Dateinamen mit der Eingabetaste ↵ aus einer Liste auswählen. Die Bezeichnungen, die in der Liste mit einem Backslash (\) enden, sind Verzeichnisse.

Program has invalid symbol table
[Programm hat ungültige Symboltabelle]
Die Symboltabelle, die am Ende Ihrer Programmdatei steht, ist nicht mehr in Ordnung. Erzeugen Sie eine neue ».EXE«-Datei und laden Sie sie erneut.

Program has no symbol table
[Programm hat keine Symboltabelle]
Das Programm, das Sie debuggen wollen, wurde korrekt geladen, aber es enthält keine Symboltabelle. Sie können die Datei immer noch im *CPU*-Fenster debuggen und die nackten Daten untersuchen, aber Sie können sich nicht auf die Symbole in Ihrem Programm beziehen. Um eine Symboltabelle in Turbo Pascal für Windows zu erzeugen, müssen Sie folgendermaßen vorgehen:

1. Wählen Sie *Options/Compiler* und aktivieren den Schalter *Debug Information*.
2. Wählen Sie *Options/Linker* und aktivieren den Schalter *Debug Info in EXE*.
3. Wählen Sie *Options/Save*, um diese Einstellungen zu sichern.
4. Compileren Sie alle Module Ihres Programms erneut.

Bei der Kommandozeilen-Version des Compilers müssen Sie die Option /V angeben.

Program linked with wrong linker version
[Programm mit falscher Linkerversion gelinkt]
Sie wollten ein Programm debuggen, das veraltete Debuggerinformationen enthält. Linken Sie Ihr Programm mit der letzten Version des Linkers oder compilieren Sie Ihre Datei mit der letzten Version von Turbo Pascal.

Program not found
[Programm nicht gefunden]
Der von Ihnen angegebene Programmname existiert nicht. Geben Sie entweder den korrekten Dateinamen an oder wählen Sie das Programm mit der Auswahlliste aus.

Register cannot be used with this operator
[Register kann nicht mit diesem Operator verwendet werden]
Sie wollten einen Befehl assemblieren, in dem ein Basis- oder Indexregister mit einem negativen Offset verwendet wird. Sie dürfen für die Basis- und Indexregister nur positive Offsets verwenden. Statt

```
INC    WORD PTR [12-BX]
```

darf nur folgender Befehl verwendet werden:

```
INC    WORD PTR [12+BX]
```

Register or displacement expected
[Register oder Offset erwartet]
Sie wollten einen Befehl assemblieren, der einen unzulässigen Ausdruck zwischen den eckigen Klammern enthält. Sie dürfen nur Registerbezeichnungen oder Offset-Angaben in den eckigen Klammern verwenden.

Repeat count not allowed
[Wiederholungsfaktor nicht erlaubt]
Sie haben einen Formatstring mit einem Wiederholungsfaktor eingegeben, aber der Wiederholungsfaktor ist in diesem Ausdruck nicht zulässig.

Run out of space for keystroke macros
[Kein Platz mehr für weitere Makros]
Bei der Aufnahme des Makros ist der Platz ausgegangen. Sie können nur bis zu 256 Tastenanschläge für alle Makros speichern.

Search expression not found
[Gesuchter Ausdruck nicht gefunden]
Der gesuchte Text oder die gesuchten Bytes wurden nicht gefunden. Die Suche fängt bei der aktuellen Cursorposition in der Datei an und verläuft nur vorwärts. Wenn Sie eine Datei ganz durchsuchen wollen, müssen Sie vor der Eingabe des Suchbefehls mit [Ctrl][PgUp] an den Anfang der Datei springen.

Source file ___ not found
[Quelldatei ___ nicht gefunden]
Turbo Debugger kann die Quelldatei für das Modul, das Sie untersuchen wollen, nicht finden. Vor der Ausgabe dieser Meldung wurde an verschiedenen Stellen gesucht:

- dort, wo der Compiler die Quelldatei gefunden hat;
- in den Verzeichnissen, die durch die Kommandozeilenoption -sd und den Befehl *Options/Path for source* festgelegt wurden;
- im aktuellen Verzeichnis;
- im Verzeichnis, in dem Turbo Debugger das Programm gefunden hat, das Sie debuggen.

Sie sollten das Verzeichnis, das die Quelldatei enthält, in die Liste der Suchverzeichnisse aufnehmen. Verwenden Sie dazu den Befehl *Options/Path for source*.

Symbol not found
[Symbol nicht gefunden]
Sie haben einen Ausdruck mit einer ungültigen Variablenbezeichnung eingegeben. Sie haben sich vielleicht vertippt oder die Variable befindet sich in einer anderen Prozedur oder Funktion als der aktuellen oder außerhalb des Gültigkeitsbereichs in einem anderen Modul.

Symbol table file not found
[Symboldatei nicht gefunden]
Die angegebene Datei mit der Symboltabelle existiert nicht. Sie können entweder eine ».TDS«- oder ».EXE«-Datei für die Symboltabelle angeben.

Syntax error
Syntaxfehler
Sie haben einen Ausdruck mit einem falschen Format angegeben. Dies ist eine allgemeine Fehlermeldung für Fälle, in denen die Fehlerursache nicht näher bestimmt werden kann.

Too many files match wildcard mask
[Zuviele Dateien mit den Jokerzeichen ausgewählt]
Sie haben eine Dateimaske mit Jokerzeichen eingegeben, mit der mehr als 100 Dateien gefunden wurden. Es können aber nicht mehr als 100 Dateien dargestellt werden.

Unexpected end of line
[Vorzeitiges Zeilenende]
Beim Auswerten eines Ausdrucks ist dessen Ende erreicht worden, bevor ein gültiger Ausdruck erkannt werden konnte. Statt

```
99 - 22 *
```

hätte es vielleicht

```
99 - 22 * 4
```

heißen sollen, und

```
SUB     AX,
```

war vielleicht wohl als

```
SUB     AX,4
```

gedacht.

Unknown character
[Unbekanntes Zeichen]
Sie haben einen Ausdruck eingegeben, der ein Zeichen enthält, das in einem Ausdruck nicht verwendet werden darf, wie zum Beispiel das umgekehrte einfache Anführungszeichen (‘).

Unknown record or structure name
[Unbekannte Record- oder Strukturbezeichnung]
Sie haben einen Ausdruck eingegeben, der eine Typumwandlung mit einer unbekannten Record-, Struktur- oder Variantenbezeichnung enthält. (In Assembler haben Strukturen einen eigenen Symbolraum, den sie nicht mit anderen Variablen teilen müssen.)

Unknown symbol
[Unbekanntes Symbol]
Sie haben einen Ausdruck mit einer ungültigen lokalen Variablen eingegeben. Entweder ist der Modulname falsch oder das Symbol oder die Zeilennummer ist nicht richtig.

Unterminated string
[String ohne Ende]
Sie haben einen String eingegeben, der nicht mit einem Anführungszeichen abgeschlossen wurde (' in Pascal). Wenn Sie einen String in Pascal-Syntax eingeben wollen, müssen Sie ein einfaches Anführungszeichen einfügen.

Value must be between 1 and 32
[Wert muß zwischen 1 und 32 liegen]
Sie haben einen ungültigen Wert für den Tabulatorabstand eingegeben. Die Tabulatoren müssen mindestens eine Spalte voneinander entfernt sein, dürfen aber nicht mehr als 32 Spalten umfassen.

Value out of range
[Wert außerhalb des Wertebereichs]
Sie haben einen Wert für eine Pascal-Variable eingegeben, die außerhalb des zulässigen Wertebereichs liegt.

Video mode not available
[Bildschirmmodus nicht verfügbar]
Sie wollten in den 43-/50-Zeilenmodus umschalten, aber Ihr Bildschirmadapter unterstützt diesen Modus nicht. Sie können den 43-/50-Zeilenmodus nur mit einem EGA- oder einem VGA-Adapter verwenden.

< Diese Seite wurde absichtlich leer gelassen>

E. Der Einsatz mit verschiedenen Sprachen

In diesem Anhang haben wir einige Tips zusammengefaßt, wie man Turbo Debugger am effektivsten mit den einzelnen Programmiersprachen einsetzen kann.

Tips für Turbo Assembler

Reine Hexadezimaldaten untersuchen

Mit Hilfe der Befehle *Data/Add Watch* und *Data/Evaluate* können Sie, wenn Sie das entsprechende Ausgabeformat wählen, auch reine Hexadezimalausgaben untersuchen. So werden zum Beispiel mit

```
[ES:DI],20m
```

die ersten 20 Bytes, die durch das Register-Paar ES:DI adressiert werden, als Hexadezimalzahlen ausgegeben.

Quellcode debuggen

Sie können Ihren Assemblerquellcode in einem *Module*-Fenster genauso bearbeiten, wie dies bei den höheren Programmiersprachen der Fall ist. Wenn Sie die Werte in den Registern sehen wollen, können Sie ein *Register*-Fenster rechts neben dem *Module*-Fenster öffnen.

Manchmal will man sowohl das CPU-Fenster als auch den Quellcode sehen. Um dies zu tun, öffnen Sie ein CPU-Fenster und wählen dann den lokalen Befehl *Mixed* im Codeausschnitt aus, bis Sie »BOTH« lesen. Auf diese Weise können Sie sowohl Ihren Quellcode als auch die Maschinensprachebefehle sehen. Beachten Sie bitte, daß Sie das CPU-Fenster mit F5 auf den ganzen Bildschirm ausdehnen

müssen, wenn Sie die einzelnen Bytes der Maschinensprachebefehle sehen wollen.

Register untersuchen und verändern

Der offensichtlich einfachste Weg, ein Register zu verändern, erfolgt über die Auswahl des Registers entweder im *CPU*-Fenster oder im *Register*-Fenster. Bei einer anderen Methode verwendet man den Befehl *Data/Evaluate/Modify*. Sie können dann eine Zuweisung eingeben, mit der der Registerinhalt verändert wird, wie zum Beispiel in

```
SI=99
```

Damit wird das SI-Register mit 99 geladen. Auf die gleiche Art und Weise können Sie ein Register auch untersuchen. Mit [Alt][D][E] AX erhalten Sie den Wert des AX-Registers.

Tips für Turbo Pascal

Der Initialisierungscode

Nachdem Sie Ihr Programm in Turbo Debugger geladen haben, zeigt der Pfeil auf das reservierte Wort begin im Hauptprogramm. Diese begin-Anweisung entspricht eigentlich einer Reihe von Initialisierungsaufrufen in allen Units, die Ihr Programm verwendet (falls Sie überhaupt einen Initialisierungscode haben). Alle Programme fangen mit einem Aufruf des Initialisierungscodes in der Unit *System* an.

Zu diesem Zeitpunkt können Sie F7 drücken, um in den ersten Initialisierungscode der ersten Unit zu gehen, in der Debug-Informationen vorliegen. Wenn Sie bis zum Ende der ersten Routine F7 verwenden, können Sie bis zur nächsten Initialisierungsroutine wandern. Eventuell kehren Sie auch gleich zum Hauptprogramm zurück und stehen dann in der ersten Anweisung.

Wenn Sie aber sofort am Anfang des Programms F8 (das Tastenkürzel für *Run/Step over*) drücken, übergehen Sie den gesamten Initialisierungscode und landen direkt in der ersten Anweisung des Hauptprogramms.

Die Exit-Prozeduren

Bevor Ihr Programm beendet wird, werden zuerst eine Reihe von Exit-Prozeduren abgearbeitet. Wenn Sie die letzte end-Anweisung des Hauptprogramms schrittweise ausführen, landen Sie jedoch nicht in den Exit-Prozeduren. Um dorthin zu gelangen, müssen Sie einen Breakpoint in jeder einzelnen Prozedur angeben, die Sie debuggen wollen.

Konstanten

Konstante Bezeichner werden von Turbo Debugger nur bei Skalaren oder typisierten Konstanten erkannt. Wenn Sie das folgende Programm

```
program Test;
const
  A = 5;
  B = Pi;
  Mtlg = 'Test';
  Grossbuchst = ['A'..'Z'];
  Ziffern : string[10] = '0123456789';
begin
  WriteLn(A);
  WriteLn(B);
  WriteLn(Mtlg);
  WriteLn('A' in Grossbuchst);
  WriteLn(Ziffern);
end.
```

debuggen, dann können Sie A (eine skalare Konstante), B (eine Gleitkommakonstante), Mtlg (ein String) und Ziffern (eine typisierte Konstante) in einem *Inspector*-Fenster untersuchen, aber Grossbuchst (eine Mengenkonstante) kann nicht untersucht werden.

Strings und Mengen auf dem Stack

Wenn Sie das *CPU*-Fenster verwenden, sollten Sie wissen, daß Turbo Pascal Zwischenergebnisse von Strings und Mengen auf den Stack legt.

Das Pluszeichen in Verbindung mit Strings sowie alle anderen Stringfunktionen bewirken, daß für die Ergebnisse dieser Operationen Speicherplatz auf dem Stack reserviert wird. Dieser Bereich wird im Stackbereich der aufrufenden Prozedur reserviert. Genauso wird für die Mengenoperatoren +, - und * ein Stack-Bereich für Zwischenergebnisse reserviert.

Geschickte Datentypumwandlungen

Die Unit *Dos* definiert das interne Datenformat für alle vordefinierten Dateitypen. Sie können diese Deklarationen verwenden, um die Daten jeder Dateivariablen zu untersuchen. Geben Sie dazu folgendes Programm ein:

```
program TypUmwandeln;
uses WinDos;
var
  TextDatei : Text;
  IntDatei : file of Integer;
begin
  Assign(TextDatei, 'Text.DTA');
  Rewrite(TextDatei);
  Assign(IntDatei, 'Int.DTA');
  Rewrite(IntDatei);
  Close(TextDatei);
  Close(IntDatei);
end.
```

Nun geben Sie die folgenden vier *Watch*-Ausdrücke ein;

```
IntDatei
TextDatei
FileRec(IntDatei)
TextRec(TextDatei)
```

In den ersten beiden Ausdrücken erhalten Sie den Status (CLOSED, OPEN, INPUT, OUTPUT) und den Namen der Datei, während Sie mit der Datentypumwandlung in den anderen beiden Ausdrücken die Namen der Felder und der Werte der Dateivariablen sehen.

Tips für das CPU-Fenster in Pascal

- Die Routinen in der Unit *System* haben keine Bezeichnungen. Wenn Sie einen CALL-Aufruf im *CPU*-Fenster beobachten, finden Sie nur eine absolute Adresse statt einer Symbolbezeichnung.
- Eine Reihe von Ein/Ausgaberoutinen, wie zum Beispiel ReadLn und WriteLn, erzeugen in Assembler mehrere Aufrufe.
- Die Überprüfung der Bereichsgrenzen, des Stacks und der Ein-/Ausgabe erfolgt in Bibliotheksroutinen.
- Eine Reihe von Operatoren (LongInt-Multiplikation, Stringverkettung usw.) sind als Bibliotheksaufrufe implementiert.
- Die Konstanten von Strings, Mengen und Fließkommazahlen befinden sich im Code-Segment, direkt vor dem Einsprungspunkt der Prozedur.

F. Glossar

Die Ausdrücke, die im Glossar zusammengestellt sind, werden in diesem Handbuch häufig verwendet. Einige davon beziehen sich auf Computer und Software allgemein, andere betreffen Turbo Debugger selbst.

Aktion	Operation, die beim Auslösen eines Breakpoints vorgenommen wird, etwa das Anhalten Ihres Programms, die Auswertung eines Ausdrucks, das Festhalten des Ergebnisses in einem Protokoll.
aktiver Ausschnitt	Alle Benutzereingaben und Befehle für ein lokales Menü beziehen sich auf diesen Ausschnitt.
aktives Fenster	Momentan ausgewähltes Fenster, in dem die Eingaben verarbeitet werden.
Array	Ein Datentyp, der aus einem oder mehreren Objekten desselben Datentyps besteht, die im Speicher hintereinanderliegen und über einen Index adressiert werden können.
ASCII	Standard-Zeichensatz für IBM PC und viele andere Computer.
Assembler	Programm, mit dem man Assemblerbefehle in hexadezimale Maschinensprachebefehle umwandeln kann.
Ausdruck	Eine Kombination von Operatoren und Operanden, die nach festgelegten Syntaxregeln angeordnet werden müssen.
Ausführung rückwärts	Die Ausführung von Programmschritten ungeschehen machen
Ausschnitt	Ein Fenster kann in Turbo Debugger aus mehreren Ausschnitten bestehen. Ausschnitte dienen der logischen Gliederung der Informationen. Zum Beispiel weist das CPU-Fenster unter anderem einen Stack-Ausschnitt und einen Registerausschnitt auf.
Auswahlfeld	Wenn mehrere Einstellungen sich gegenseitig ausschließen, wie z.B. schwarzweiß und Farbe, werden diese Einstellungen in einem Dialogfenster in Auswahlfeldern vorgenommen.

automatische Variable — In C hat eine Variable dieses Typs nur innerhalb des Funktionsblocks Gültigkeit; ihr Wert wird auf dem Stack gespeichert.

Befehlsfeld — Ein Feld eines Dialogfensters, das nach Anklicken mit der Maus oder Aktivierung über die Tastatur sofort einen bestimmten Vorgang auslöst, z.B. Abbruch des Dialogs durch *Cancel*.

Bezugsrahmen — Symbole können in einem Programm sehr unterschiedliche Gültigkeitsbereiche haben. Während lokale Variablen nur in der aktuellen Funktion existieren, darf man auf statische Variablen nur innerhalb eines Moduls zugreifen, während auf globale Symbole im gesamten Programm zugegriffen werden kann. Zu jedem Zeitpunkt der Programmausführung ist also ein bestimmter Bezugsrahmen gegeben, in dem bestimmte Symbole entweder verfügbar oder nicht verfügbar sind. Mit Hilfe einer ausdrücklichen Bezugsvorgabe kann man bei Turbo Debugger auch auf Symbole zugreifen, die an der aktuellen Programmposition an sich nicht verfügbar wären.

Breakpoint — Haltepunkt, an dem das Programm angehalten wird. Der Breakpoint kann bedingt ausgelöst werden, indem man ihn mit Bedingungen und Durchlaufzählern verknüpft.

C-Ausdruck — Ausdruck, wie er in C verwendet werden darf.

CPU — Zentraleinheit des Computers, in PCs die 80x86-Prozessoren. Eine CPU besteht aus Registern, Flags, einer Recheneinheit für logische und arithmetische Operationen, Speicheranschlüssen sowie einer Reihe von Ein/Ausgabekanälen.

CPU-Flag — Ein Statusbit der CPU, durch das nachfolgende Operationen beeinflußt werden können, z.B. bedingte Sprungbefehle.

CPU-Register — Schnelle Speicherstelle in der CPU für arithmetische und logische Operationen. Die Register des 8086 sind: AX, BX, CX, DX, SI, DI, SP, BP, CS, DS, ES und SS.

CS:IP — Aktuelle Programmposition, bestehend aus Code-Segment und Programmzeiger.

Datentypvorgabe — Ausdrückliche Umwandlung eines Ausdrucks in einen anderen Datentyp, in C zum Beispiel `(char *)`.

Dialogfenster — Fenster, in dem der Benutzer seine Eingabe macht. Die letzten Eingaben werden in einer History-Liste gespeichert, mit deren Hilfe man ohne viel Tipparbeit Eingaben wiederholen kann.

Disassembler — Ein Programm, mit dem man Maschinensprachebefehle in lesbaren Assemblercode umwandeln kann. In Turbo Debug-

	ger werden im Code-Ausschnitt des CPU-Fensters alle Befehle automatisch disassembliert.
Fenster	Abgetrennter Bildschirmbereich, welcher meist einer bestimmten Aufgabe zugeordnet ist. Fenster können überlappend oder nicht überlappend angeordnet werden.
globaler Breakpoint	Breakpoint, der bei jedem einzelnen Befehl ausgelöst werden kann.
History-Liste	Liste aller vorher erfolgten Eingaben in einem Eingabefeld, die für neue Eingabe übernommen oder modifiziert werden können.
Inspector-Fenster	Fenster zum Untersuchen von Variablen und Datenstrukturen.
Jokerzeichen	Platzhalter für beliebige Zeichen in einem Suchstring oder einem DOS-Dateinamen.
Konfigurationsdatei	Datei, in der die Standardeinstellungen für ein Programm gespeichert werden. Dazu zählen zum Beispiel Position, Größe und Farbeinstellung eines Fensters. Jedesmal, wenn das Programm gestartet wird, werden zuerst die Standardeinstellungen aus der Konfigurationsdatei übernommen, die anschließend meist noch mit Hilfe von Kommandozeilenoptionen auch im Programm selbst verändert werden können.
lokales Menü	Jeder Ausschnitt eines Fensters hat ein lokales Menü, in dem eine Reihe von Befehlen verwendet werden, mit denen man den Inhalt des jeweiligen Fensters am besten manipulieren kann.
Markierungsfeld	Ein Feld in einem Dialogfenster, mit dem man eine Einstellung ein- oder ausschalten kann.
Menge	Ungeordnete Gruppe von skalaren Elementen. Siehe `enum` in C oder `set` in Pascal.
Menüleiste	In der obersten Zeile des Bildschirms befindet sich eine Menüleiste mit sogenannten Pulldown-Menüs, die bei der Auswahl einer Menüoption erscheinen (Alt und der erste Buchstabe des Menüs). Diese Befehle sind jederzeit verfügbar.
Operand	Datenobjekt, mit dem eine Operation mit Hilfe von Operatoren ausgeführt wird. Im folgenden Beispiel werden die Operanden x und 4 in der Speicherstelle y addiert: `y = x + 4`
Operator	Meist symbolische Kurzform für eine Operation, wie zum Beispiel die Addition (+) oder die Multiplikation (*).

PATH	Environment-Variable unter DOS, in der alle Verzeichnisse angegeben werden, die bei einem Befehlsaufruf nach der entsprechenden Programmdatei durchsucht werden.
Popup-Menü	Ein Popup-Menü erscheint, wenn es die Situation erfordert. Die lokalen Menüs der Turbo-Debugger-Fenster sind Popup-Menüs.
Pulldown-Menü	Menü, das von einer Menüleiste »heruntergezogen« wird. (Die Menüs in der Menüleiste sind typische Pulldown-Menüs.)
Seiteneffekt	Ein Ausdruck, der den Wert einer Variablen oder einer Speicherstelle ändert.
Skalar	Ein grundlegender Datentyp mit abzählbaren Werten, wie Byte, Integer oder Boolean in Pascal oder `char`, `int` oder `long` in C bzw. C++.
Stack	Datenbereich mit einer LIFO-Struktur, auf dem lokale Variablen, Parameter und Adressen von Prozeduren untergebracht werden.
Symbol	Bezeichnung einer Variablen, Konstanten, Prozedur oder Funktion.
Tracen	Programm Schritt für Schritt ausführen.
Watches-Fenster	Fenster, in dem Variablen laufend überwacht werden können.

Index

A
Add comment (Protokoll) 99
Add Watch
  Data 80
Adressen 112
Aktion 91
  festlegen 94
Aktion (Glossar) 289
Aktionsschalter 17
aktiver Ausschnitt (Glossar) 289
Alt-Funktionstasten 29, 32
Always 56
Always (Breakpoints/Set options) 95
Animate 69
Anweisung
  .STACK 198
  PROC 198
Argumente 2, 70
  ändern 75
Arithmetikprozessor 163
Array (Glossar) 289
ASCII (Glossar) 289
ASCII-Dateien 107
Assemble
  CPU-Fenster (Code) 151
Assembler 145, 158
  Befehle 241
  Datentypen 242
  gemischte Darstellung 150, 249
  Inline- 241
  Syntax für Ausdrücke 253
Assembler (Glossar) 289
Assembler-Modus 52
Ausdruck (Glossar) 289
Ausdrücke 78, 111
  Auswahl der Programmiersprache 111
  auswerten 102
  Formatangaben 120
  in Assembler 118
  in Breakpoints 93
  in Pascal 116
Ausführung rückwärts (Glossar) 289
Ausgabefenster 26
Ausschnitt
  wechseln 29
Ausschnitt (Glossar) 289
Auswahl
  sukzessive 21
Auswahlbalken 29
Auswahlfeld (Glossar) 289
automatische Variable (Glossar) 290

B
Back trace 69
Base segment
  CPU-Fenster (Daten) 155
Bedingungen 91, 95
Beep on error 252
Befehle
  CALL 197
  IMUL 200
  MUL 200
  RET 197, 198
Befehlsausschnitt 71
  lokales Menü (Execution history) 72
Befehlsfeld (Glossar) 290
Befehlsübersicht 167
Beispielprogramm 35
Beispielprogramme 35
Benutzerbildschirm 26
Benutzeroberfläche
  Dialogfenster 20
  Fenster 22
  History-Listen 20
  lokale Menüs 18
  lokales Menü 19
  Menüleiste 15
  Tastenkürzel 32

Bereiche untersuchen 89
Bezeichner
  automatische Ergänzung 21
Bildlaufleiste 27
Bildschirm
  »Schnee« 250
  Anpassung 247
  direkt in Speicher schreiben 250
  zweiter 52
Bildschirmfarben 246
  Anpassung 248
Bildschirmmodus 254
Bildschirmparameter 248
Bildschirmumschaltung 51, 248
  Always 56
  Smart 55
Block-Befehle 156
Both (Integer format) 56
Break (Breakpoints/Set options) 95
Breakpoint
  globaler (Glossar) 291
Breakpoint (Glossar) 290
Breakpoint at 66
Breakpoints 91
  Aktion 91
  alle löschen 93
  Ausdrücke 96
  Ausdrücke abfragen 93
  ausschalten 96
  Bedingungen 91, 95
  Beispiel 40
  definiert setzen 92
  Durchlaufzähler 96
  einschalten 96
  entfernen 97
  globale 101
  globale Speicherüberwachung 93
  Gültigkeitsbereich 93
  hinzufügen 96
  Konzepte 91
  löschen 97
  mit Bedingungen 100
  normale 99
  Programm bis zum B. laufen lassen 69
  Umschalten 92
  untersuchen 97
Breakpoints (Menü) 92
Breakpoints (View) 94
Breakpoints mit Durchlaufzähler 100
Breakpoints-Fenster 23, 94
  lokales Menü 94, 172
Breakpoints-Menü
  Übersicht 170
Byte-Folgen 115

C
CALL 197
Caller
  CPU-Fenster (Code) 149
Carry-Flag 200
Change 14
  CPU-Fenster (Daten) 155
  CPU-Fenster (Register) 153
  CPU-Fenster (Stack) 158
  globale Variablen 61
  Inspector 89
  Instanzen 128
  lokale Variablen 62
  Numeric processor (Register) 165
  Watches 83
Changed memory (Breakpoints/Set options) 95
Changed memory global
  Breakpoints 93
Close log file 99
CMPS 199
Code-Ausschnitt 147
  lokales Menü 148, 173
Control-Break 66, 73
Control-Funktionstasten 29, 32
CPU (Glossar) 290
CPU-Fenster 24, 146
  Datenausschnitt 153
  lokale Menüs 148, 173f.
  Registerausschnitt 152
  Stackausschnitt 157
  Statusausschnitt 152
CPU-Flag (Glossar) 290
CPU-Register 242
CPU-Register (Glossar) 290
Create (Options/Macros) 55
CS:IP 147, 150
CS:IP (Glossar) 290
Ctrl-Break 73
Cursor-Bewegungen 29

D
Data-Menü 78
  Übersicht 171
Data/Function return 80
Dateien 103, 107
Dateifenster 107
Dateimaske 183

Daten
  ändern 44
  untersuchen 42
Daten untersuchen 78
Datenausschnitt
  CPU-Fenster 153
  lokales Menü 154, 173
  Zahlendarstellung 153
Datentypen
  in Assembler 242
Datentypvorgabe (Glossar) 290
Debuggen
  auf Assembler-Ebene 145
  Beispiel 203
  Einführung 35
  in der Praxis 187
  Methodik 11
Debugger
  Einsatzgebiet 13
Debugger starten
  Beispiel 36
Decimal 56
Decrement
  CPU-Fenster (Register) 152
Delete All
  Breakpoints 93, 97
  Watches 83
Delete All (Options / Makro) 55
Demo-Programme 35
Descend
  Inspector 90
  Instanzen 128
Dialogfenster 15, 16f., 20, 257
  akustisches Signal 252
  Anpassung 247
  Befehle 182
  Funktionsweise ändern 252
  Hotkeys 17
  im Evaluate-Fenster 79
  Mausbedienung 17
Dialogfenster (Glossar) 290
Dienstprogramme 8
  TDSTRIP 9
  TDUMP 9
  TDWINST 9
Direction-Flag 199
Disassembler
  Code-Ausschnitt 148
Disassembler (Glossar) 290
Display as
  CPU-Fenster (Daten) 156
  File-Fenster 109
Display options (Options) 55
Display swapping 55
DOS Version
  ermitteln 65
Dump-Fenster 24, 160, 175
Durchlaufzähler 96

E
Edit
  Watches 82
EGA
  43-Zeilen-Modus 56, 249
Ein-/Ausgabeports 151
Einführung 35
Eingabefelder 17
  Editiermöglichkeiten 182
Einzelschritt 70
EMS 65
  für Execution history 71
Erase log 99
Exception 66
Execute (Breakpoints/Set options) 95
Execute to 69
Execution history 25, 71, 175
  Full history 72
  Reverse Execute 72
Expression true (Breakpoints/Set options) 96
Expression true global
  Breakpoints 93

F
Far code
  CPU-Fenster (Daten) 155
Farbbildschirm 254
Fehlermeldungen 257
  andere 266
  schwerwiegende 265
Fehlermöglichkeiten 189
Fehlerquellen
  Assembler 197
Fehlersuche 11
Fenster 22
  aktives 26
  aktives (Glossar) 289
  Anordnung ändern 30
  Anordnung speichern 30
  Anpassung 246
  Ausgabe- 26
  Bedienung 28
  Beispielprogramme 38
  Bestandteile 27

Bildlaufleiste 27
Breakpoints 23, 94
CPU 24, 146
Dump 24, 160
Execution history 25, 71
File 24, 107
Hierarchy 25, 123
Inspector 26, 78, 83
Konfiguration 31
Log 23, 97
mehrfache 26
Module 23, 104
nicht-überlappende 250
Numeric processor 25, 163
Rahmen 27
Registers 25, 161
schließen 30
Schließfeld 28
Stack 23
Symbolfeld 28
Titel 27
Variables 24, 60
Vergrößerungsfeld 27
View-Menü 22
Watches 23, 81
wechseln 28
wiederherstellen 30
Windows Messages 25
zoomen 30
Zoomfeld 28
Fenster (Glossar) 291
File
File-Fenster 110
Module-Fenster 106
File / Get Info 65
File-Fenster 24, 107
lokales Menü 108, 175
File/Open 75
Flagausschnitt
lokales Menü 174
Flags 147
zerstörte 201
Follow
CPU-Fenster (Code) 149
CPU-Fenster (Daten) 155
CPU-Fenster (Stack) 158
Formatsteuerung 120
Full history
Execution history 72
Full trace history 253
Function return
Data 80
Funktionen
in Pascal aufrufen 118
Funktionstasten 32, 167
mit Alt 32
mit Control 32

G
Get Info 65
Gleitkomma-Emulator 163
Global breakpoint at 67
Globalausschnitt
lokales Menü 179
globale Breakpoints 101, 102
Go to cursor 68
Goto
CPU-Fenster (Code) 148
CPU-Fenster (Daten) 154
CPU-Fenster (Stack) 157
File-Fenster 108
Module-Fenster 107
Groß- und Kleinschreibung 53, 254
Gültigkeitsbereich 113
impliziter 115
in Breakpoint-Ausdrücken 93
Vorgabe 113

H
Hardware-Voraussetzungen 1
Help 31
HELPME!.TDW 8
Hex 56
Hex-Dateien 107
Hexadezimalausgaben 285
Hierarchiebaum 25, 124
Inspect 126
Hierarchy
Inspector-Fenster 127
Instanzen 129
Objekttypen 126
Hierarchy-Fenster 25, 123
Übersicht 178
Hilfestellungen 31
Beispiel 37
in Menüs 33
zum Aufruf von TDW 52
Hilfesystem 31
History-Liste 20
Länge 252
History-Liste (Glossar) 291
Hotkeys 167
benutzerdefinierte 22

Dialogfenster 17
Hotline 4

I
I/O
CPU-Fenster (Code) 151
IMUL 200
In byte
CPU-Fenster (Code) 151
Increment
CPU-Fenster (Register) 152
Inline-Assembler 158
Input prompting 252
Inspect 13, 77, 78
Breakpoints 97
Data 78
Execution history 72
globale Variablen 61
Hierarchiebaum 126
Inspector 89
Instanzen 128
Klassen 124
lokale Variablen 62
Module-Fenster 105
Objekttyp/Klasse 124
Objekttypen 124, 126
Stack 64
Watches 83
Inspector-Fenster 26, 61, 77, 78, 83
(Glossar) 291
Change 89
Daten zurückverfolgen 90
Descend 90
Inspect 89
lokales Menü 89, 179
neue Variablen untersuchen 90
neuer Datentyp 90
neues eröffnen 89
New expression 90
Range 89
Type Cast 90
Installationsprogramm 245
Instruction trace 70
Integer-Darstellung 56, 249
Interrupt (Meldung) 66
Interrupt-Aufrufe
bei Programmaufzeichnung 72
Interrupthandler 201

J
Jokerzeichen 183
Jokerzeichen (Glossar) 291

K
Klassen
Gültigkeitsbereich 114
Hierarchie untersuchen 123
Hierarchiebaum 124
Inspect 124
Inspector-Fenster 125
Liste aller Typen 124
Mehrfachvererbung 124
Show Inherited 126
Tree 124
Klasseninstanzen
untersuchen 127
Kommandozeile
Optionen 239
Syntax 239
Kommandozeilenaufruf 48
Kommandozeilenoption
-? 52
Kommandozeilenoptionen 51, 239
-do 52
-ds 52
-h 52
-l 52
-p 52
-s 53
-sc 53
-sd 53
-t 53
Groß- und Kleinschreibung 53
Mausunterstützung 52
Quellcode 53
Startverzeichnis 53
Suchverzeichnisse 53
Kommandozeilensyntax 49
Kommandzeilenaufruf
Beispiele 49
Kommentare
in Log-Datei 99
Konfiguration
Fenster 31
Konfigurationsdatei 51, 54
Konfigurationsdatei (Glossar) 291
Konstanten
in Assembler 119
in Pascal 117
Kontroll-Ausschnitt
Numeric processor 165
Kontrollausschnitt (Koprozessor) 165
Koprozessor 163
Befehle 243

Kontrollausschnitt 165
Statusausschnitt 166

L
Language (Options) 54
LCD-Bildschirm 254
Lieferumfang 7
Line
Module-Fenster 106
Listen 17
Listenausschnitte 181
Loaded 66
Locals
Variables 61
Log (Breakpoints/Set options) 95
Log-Datei
Kommentare einfügen 99
öffnen 98
schließen 99
Log-Fenster 23, 97
Länge 251
lokales Menü 98, 175
löschen 99
Lokalausschnitt
lokales Menü 179
lokale Befehle
Übersicht 172
lokale Menüs 18f., 172f., 175, 177f.
Befehle 19
Control-Tastenkürzel 32
lokales Menü (Glossar) 291
Löschen
Protokoll 99
von Breakpoints 93

M
Macros (Options) 54
Macros / Delete All (Options) 55
Macros/Remove (Options) 55
Makro
Aufnahme 22
Makros 54
beenden 55
Erstellung 22
Erstellung beenden 55
erzeugen 55
Löschen 55
MANUAL.TDW 8
Markierungsfeld (Glossar) 291
Mausunterstützung 253
Kommandozeilenoptionen 52
Mehrfachvererbung 123f.
Meldungen
Fehlermeldungen 266ff.
Dialogfenster 257
Menge (Glossar) 291
Menü 67
Breakpoints 92
Data 78
Data/Add Watch 80
Data/Evaluate/Modify 78
File (Übersicht) 169
File / Get Info 65
File / Quit 59
Options 54
Options / Display options 55
Options / Environment Tab Size 56
Options / Integer format 56
Options / Language 54
Options / Macros / Delete All 55
Options / Path for source 57
Options / Restore options 59
Options / Save options 57
Options / Screen lines 56
Options / Stop Recording 55
Options/Macros 54
Options/Macros/Create 55
Options/Macros/Remove 55
Run / Animate 69
Run / Backtrace 69
Run / Execute to 69
Run / Go to Cursor 68
Run / Run 67
Run / Step over 68
Run / Trace into 68
Run / Until return 69
Run /Instruction trace 70
Run/Program reset 70
Stackausschnitt 157
System (Übersicht) 169
View / Stack 61, 63
View / Variables 61
View/Breakpoints 94
View/CPU 146
View/File 107
View/Module 104
View/Watches 81
Menübaum 183
Menüleiste 15, 19, 37
Befehle 169
Menüleiste (Glossar) 291
Menüs 15
allgemeine Menüs (Menüleiste) 19
Anpassung 247

lokale 18
lokale Befehle (Übersicht) 172
Mausbedienung 16
Tastaturbedienung 16
Methoden
untersuchen 128
Methods
Instanzen 128
Mixed
CPU-Fenster 148
CPU-Fenster (Code) 150
Module (View) 104
Module-Fenster 23, 104f.
lokales Menü 105, 176
Modulliste 99
Monochrome-Bildschirm 254
Mouse enabled 253
MUL 200
Multiplikation 200

N
Nachkomme
im Objektbaum 124
Namensergänzung
automatische 21
Near code
CPU-Fenster (Daten) 155
New CS:IP
CPU-Fenster (Code) 150
New expression 90
Instanzen 129
Next
CPU-Fenster (Daten) 154
File-Fenster 109
Module-Fenster 106
No program loaded 65
Numeric Processor 163
Kontroll-Ausschnitt 165
lokale Menüs 177
Registerausschnitt 164
Statusausschnitt 166
Numeric processor-Fenster 25

O
Objekte
Gültigkeitsbereich 114
Hierarchie untersuchen 123
Objekthierarchie
Darstellung beim Debuggen 129
Objektinstanzen
untersuchen 127
objektorientiertes Pascal 123
Objekttyp-Inspector-Fenster
lokales Menü 126
Objekttypen
Inspect 124, 126
Inspector-Fenster 125
Liste aller 124
Hierarchy 126
Show inherited 126
Tree 124
Offset to data
CPU-Fenster (Daten) 155
Open log file 98
Operand (Glossar) 291
Operanden
im Speicher 201
konstante 201
vertauschte 198
Operator (Glossar) 291
Operatoren
in Assembler 119
in Pascal 117
Optionen 51
Aufruf von Turbo Debugger 239
speichern 57
Utility-Programme 9
Optionen laden 59
Options (Menü) 54
Options-Menü
Übersicht 171
Origin 64
CPU-Fenster (Code) 149
CPU-Fenster (Stack) 157
Module-Fenster 107
Originaldisketten 7
Out byte
CPU-Fenster (Code) 151

P
Parameter 51
Parent yes 124
Parser
Syntax 253
Pascal
objektorientiert 123
Syntax für Ausdrücke 253
Pass count (Breakpoints/Set options) 96
PATH (Glossar) 292
Path for source 57
Planungsphase 202
Popup-Menü (Glossar) 292
Popup-Menüs 16, 18
Ports 151

Probleme 4
Program reset 70
Programm
  akuelle Position 150
  Argumente 70, 75
  Aufzeichnung 71
  beenden 36, 73
  Design 202
  Einzelschritt 68
  erneut laden 74
  laden 75
  laufen lassen 67
  Neustart 36
  Quelltext untersuchen 103
  rückwärts ausführen 69
  schrittweise ausführen 68
  starten 67
  unterbrechen 73, 102
  zurücksetzen 70
Programmablauf 67
Programmausführung 59
  rückwärts 69
Programmbeispiele
  fehlerhafte 203
Programmdateien 7
Programme
  Vorbereitung zum Debuggen 47
Programmende 59
Programmiersprache 111, 253
Programmschritte
  in Execution history 71
Programmstatus 65
Protokoll 97
  Anzahl der Zeilen 251
  Datei öffnen 98
  Datei schließen 99
  einschalten 99
  von Variablenwerten 102
Prozeduren
  in Pascal aufrufen 118
Prozessor
  Anzeige 147
Pulldown-Menü (Glossar) 292
Pulldown-Menüs 16

Q
Quellcode
  Optionen 53
  Verzeichnisse 104
Quelltextdarstellung 249
Quelltexte
  untersuchen 103
Quelltextverzeichnisse 53

R
Range
  Inspector 89
  Instanzen 128
Rangfolge
  Assembler 118
  Pascal 117
Read word
  CPU-Fenster (Code) 151
README 8
Register
  Koprozessor 242
  löschen 198
  retten 201
Registerausschnitt
  CPU-Fenster 152
  lokales Menü 152, 174
  Numeric processor 164
Registers-Fenster 25, 161
Remove
  Breakpoints 97
  Watches 82
Remove (Options/Macros) 55
Restore options 59
RET 197, 198
Reverse Execute
  Execution history 72
Rücksprung 197
rückwärts
  Programmausführung 69
Run (Menü) 67
  Übersicht 170
Run / Backtrace 69
Run/Arguments 75
Run/Program reset 70

S
Save configuration file 255
Save options 57
Schaltfelder 17
Schließfeld 28
Schwarzweiß-Bildschirm 254
Screen lines 56
Scrollen 27
Search
  CPU-Fenster (Code) 149
  CPU-Fenster (Daten) 154
  File-Fenster 109
  Module-Fenster 106
Segment
  STACK 198
Segment:offset to data
  CPU-Fenster (Daten) 155

Segmente
  Überlauf 201
Seiteneffekt (Glossar) 292
Seiteneffekte
  Multiplikationen 200
Set options (Breakpoints) 94
Show
  lokale Variablen 63
Show inherited 127
  Instanzen 128
  Klassen 126
  Objekttypen 126
Skalar (Glossar) 292
Smart 55
Software-Voraussetzungen 1
Source debugging 253
Speicherplatz
  Belegung 65
Speicherüberwachung 93, 102
Sprungbefehle
  bedingte 198
Stack 198
  Fenster 63
  lokale Variablen 64
  lokales Menü 64
  Position 64
  untersuchen 64
Stack (Glossar) 292
Stack-Fenster 23, 63
  lokales Menü 64, 178
Stackausschnitt
  CPU-Fenster 157
  lokales Menü 157, 174
Startbildschirm
  Assembler 52
Startverzeichnisse
  Kommandozeilenoptionen 53
Statusanzeige 31
Statusausschnitt 147
  CPU-Fenster 152
  lokales Menü 153
  Numeric processor 166
Statusausschnitt (Koprozessor) 166
Statuszeile 37
Step 13
Step (Meldung) 66
Step over 68
Stop recording (Options / Makro) 55
Stopped at 65
Stringbefehle 160
  Byte in Word-Operand umwandeln 200
  CMPS 199
  Direction-Flag 199
  Operanden 200
  Standard-Segmente 199
  veränderte Register 200
  Vorgaben 200
  Wiederholungen 199
  Zeigervorlauf 199
Strings
  in Pascal 117
Suche
  sukzessive 29
Suchen 183
  in Dateien 109
Suchverzeichnisse 50, 57, 252
  Kommandozeilenoptionen 53
sukzessive Auswahl 21
Symbol (Glossar) 292
Symbole
  Groß- und Kleinschreibung 254
  Gültigkeitsbereich 113
  in Assembler 119
Symbolfeld 28

T
Tabulator 56
Tabulatorbreite 250
Taschenrechner 80
Taschenrechnerfunktion 111
Tastaturbefehle 31
Tastenkürzel
  benutzerdefinierte 22
TDCONFIG.TDW 51, 54
TDDEMO.EXE 36
TDDEMO.PAS 36
TDSTRIP 9
TDUMP 9
TDW 51
  Kommandozeilenoptionen 51
  Optionen laden 59
  Optionen speichern 57
TDWINST 9, 245
  43/50 Line Mode Startup 249
  Aufruf 245
  Bildschirmfarben 246
  Bildschirmparameter 248
  Display swapping 248
  Fast screen update 250
  Floating precision 251
  Ignore symbol case 254
  Integer format 249
  Kommandozeilenoptionen
  und Dialogfenster 254

Konfiguration speichern 255
Language 253
Log list length 251
Maximum tiled watch 250
Optionen 252
Options 252
Permit 43/50 Lines 250
Quit 256
Range inspect 251
Startup Display 249
Tab size 250
User screen updating 250
verlassen 256
Terminated, exit code 67
Textausschnitte 181
Toggle
Breakpoints 92
CPU-Fenster (Status) 153
Numeric processor (Kontroll-Bits) 165
Numeric processor (Status-Bits) 166
Trace (Meldung) 66
Trace into 68
Tracen 13, 68
bei Programmaufzeichnung 72
Tracen (Glossar) 292
Tracepoint 91
Tree
Klassen 124
Objekttypen 124
Turbo Assembler
Arrays untersuchen 88
Fehlermöglichkeiten 197
Quellcode debuggen 285
Register 286
Skalare untersuchen 86
Strukturen untersuchen 88
Tips 285
Varianten untersuchen 88
Zeiger untersuchen 87
Turbo Debugger 7
andere Fehlermeldungen 266
Anpassung 245
Aufruf 47, 239
Dienstprogramme 8
Disketten 7
Fehlermeldungen 265ff.
Grenzen 14
Kommandozeilenaufruf 48
Kommandozeilenoptionen 51
Konfigurationsdatei 54
Menüs 183
Optionen 239
Programmdateien 7
Programmende 59
Vorzüge 15
Turbo Pascal
Arrays untersuchen 85
Beispielprogramm 39
CPU-Fenster 288
Datentypumwandlungen 288
Exit-Prozeduren 287
Fehlermöglichkeiten 191
Funktionen untersuchen 86
Initialisierung 286
Konstanten 287
Mengen 287
Prozeduren untersuchen 86
Records untersuchen 86
Skalare untersuchen 84
Strings 287
Tips 286, 287
Zeiger untersuchen 84
Type Cast 90
Instanzen 129

U
Unterbrechung
des Programms 73
Unterprogramme 198
Until return 69
Utility-Programme
Optionen 9

V
Variablen 60
lokale anzeigen 63
globale 61
globale ändern 61
globale überwachen 62
lokale 61
lokale ändern 62
lokale überwachen 63
lokale untersuchen 62
protokollieren 102
Variablenfenster 24, 60
globaler Ausschnitt 61
lokaler Ausschnitt 61
lokales Menü 62, 178f.
statischer Ausschnitt 61
Vererbung
Darstellung 126f.
Vergrößerungsfeld 27
Verzeichnisse
der Quelldateien 57, 104, 252

Turbo- 252
VGA
50-Zeilen-Modus 56, 249
View 13
CPU 146
Fenster 22
Hierarchy 123
Menü
Module 104
Stack 63
Übersicht 169
Variables 61
Watches 81
Vorfahr im Objektbaum 124

W
Warenzeichen 4
Watch 14
globale Variablen 62
lokale Variablen 63
Module-Fenster 105
Watches 82
Watches
Change 83
Delete All 83
Edit 82
Inspect 83
Remove 82
View 81
Watches-Fenster 23, 81
Lokales Menü 82, 179
Watches-Fenster (Glossar) 292
Watchpoint 91
Werte ändern 89
Window message breakpoint at 67
Windows
Rückkehr 59
Windows Messages 25
lokale Menüs 176f.
Windows-Info Fenster
anzeigen 99
Write word
CPU-Fenster (Code) 151

Z
Zahlendarstellung 153
Zahlenformat
in Pascal 117
Zeiger
zurückverfolgen 90
Zero
CPU-Fenster (Register) 152
Numeric processor (Register) 165
Zoomfeld 28

BENUTZER-
HANDBUCH

TURBO DEBUGGER®
für Windows

BORLAND

BORLAND GMBH, LINDWURMSTRASSE 88, POSTFACH 15 03 40, D-8000 MÜNCHEN 2, TELEFON 0 89/7 20 10-0, TELEFAX 0 89/77 93 39, TELEX 5 212 637 mcm d,
CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001. (408) 438-530. OFFICES IN:
AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN AND THE UNITED KINDOM * ARTIKEL-NR. 112306